



Chapter Four: Loops

Chapter Goals

- ใช้ `while`, `for` และ `do loops` ให้เป็น
- หลีกเลี่ยง loops รันไม่รู้จบ
- หลีกเลี่ยงข้อผิดพลาดเรื่องขอบที่มันจะคลาดเคลื่อนไปหนึ่ง (off-by-one errors)
- เข้าใจ loop ซ้อน loop (nested loops)
- เขียนโปรแกรมที่อ่านและประมวลผล data sets

What Is the Purpose of a Loop?

Loop เป็น statement ที่ใช้เพื่อ:

ทำ statement หนึ่ง หรือ หลาย statement ซ้ำ ๆ กัน
จนกว่าจะถึงเป้าหมาย

บางครั้ง ก็ไม่มี statement ไหนถูกทำเลย
—ถ้ามันเป็นวิธีที่จะตอบโจทย์ของเรา

The Three Loops in C

Java มี 3 looping statements:

while

for

do

The while Loop



In a particle accelerator, subatomic particles traverse a loop-shaped tunnel multiple times, gaining speed. Similarly, in computer science, statements in a loop are executed while a condition is true.

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop



while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop

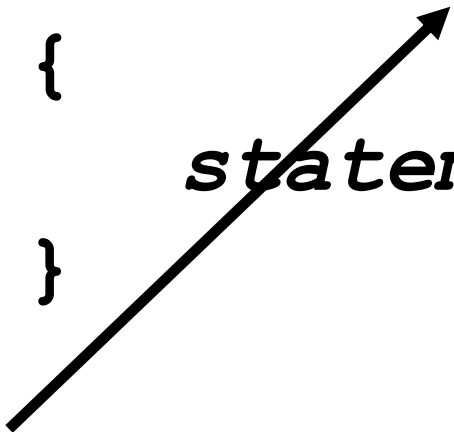


while statement จะทำ statements
จนกว่าเงื่อนไขจะเป็น false

The while Loop

```
while (เงื่อนไข)
{
    statements
}
```

ตรวจสอบเงื่อนไข
(เหมือนใน if statement)

A black arrow points from the Thai text 'ตรวจสอบเงื่อนไข (เหมือนใน if statement)' to the Thai text '(เงื่อนไข)' in the while loop syntax.

The while Loop

```
while (condition)  
{  
    statements  
}
```

statements จะถูกทำซ้ำ ๆ จนกว่าเงื่อนไขจะเป็นเท็จ

ใช้ Loop ในการแก้ปัญหาการลงทุน

An investment problem:

เริ่มต้นที่ \$10,000, เราต้องฝากเงินในธนาคารกี่ปีถึงจะถึง \$20,000?

The algorithm:

1. เริ่มที่กำหนดตัวแปร **year** เป็น 0 และ ตัวแปร **balance** เป็น \$10,000.
2. ทำขั้นตอนต่อไปนี้ซ้ำไปเรื่อย ๆ
ขณะที่ (**while**) จำนวนเงินฝาก น้อยกว่า \$20,000:
 - Increment ค่าในตัวแปร **year**.
 - คำนวณจำนวนดอกเบี้ย 5% ของจำนวนเงินฝากทั้งหมด.
 - เพิ่มดอกเบี้ยเข้าไปในจำนวนเงินฝาก.
3. ได้คำตอบเป็นปีสุดท้ายหลังจากการทำซ้ำในข้อ 2.



ใช้ Loop ในการแก้ปัญหาการลงทุน

2. ทำขั้นตอนต่อไปนี้อย่างซ้ำไปเรื่อย ๆ ขณะที่ (while) จำนวนเงินฝาก น้อยกว่า \$20,000:

“ทำซ้ำ .. ขณะที่” ในปัญหาของเรา บ่งบอกว่าเราต้องการ Loop.
เพื่อที่จะคำนวณจำนวนปีที่จำนวนเงินในธนาคารถึงเป้าหมาย, เราต้องทำการคูณและการบวกซ้ำกัน
ไปเรื่อย ๆ

ใช้ Loop ในการแก้ปัญหาการลงทุน

คำสั่งที่ถูกควบคุม (หรือถูกทำซ้ำ ๆ) ได้แก่:-

- Increment ค่าในตัวแปร **year**
- คำนวณค่า ดอกเบี้ย (**interest**)
- Update ค่าจำนวนเงินฝาก (**balance**) โดยการบวกค่าดอกเบี้ย

```
year++;
```

```
double interest = balance * RATE / 100;
```

```
balance = balance + interest;
```

Using a Loop to Solve the Investment Problem.

เงื่อนไข, ที่จะบอกว่าออกจาก **loop** ได้, คือการ
เปรียบเทียบข้างล่าง:

(balance < TARGET)

Using a Loop to Solve the Investment Problem.

Here is the complete **while** statement:

```
double interest;  
while (balance < TARGET)  
{  
    year++;  
    interest = balance * RATE / 100;  
    balance = balance + interest;  
}
```

The while Statement

This variable is defined outside the loop and updated in the loop.

If the condition never becomes false, an infinite loop occurs.

```
double balance = 0;
.  
.  
.  
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Beware of "off-by-one" errors in the loop condition.

Don't put a semicolon here!

This variable is created in each loop iteration.

These statements are executed while the condition is true.

Lining up braces is a good idea.

Braces are not required if the body contains a single statement, but it's good to always use them.

The Complete Investment Program

```
public static void main(String [] args)
```

```
ch04/doublinv.cpp
```

```
{
```

```
    final double RATE = 5;
```

```
    final double INITIAL_BALANCE = 10000;
```

```
    final double TARGET = 2 * INITIAL_BALANCE;
```

```
    double balance = INITIAL_BALANCE, interest;
```

```
    int year = 0;
```

```
    while (balance < TARGET)
```

```
    {
```

```
        year++;
```

```
        interest = balance * RATE / 100;
```

```
        balance = balance + interest;
```

```
    }
```

```
    System.out.println("Investment doubled in %d years" +  
                        year);
```

```
}
```

Program Run

Check the loop condition

balance = 10000

year = 0

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

The condition is true

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Execute the statements in the loop

balance = 10000

year = 0

interest = ?

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is true


Execute the statements in the loop

balance = 10000

year = 1

interest = ?

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```



Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is true

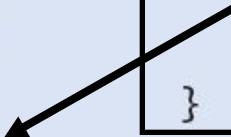
Execute the statements in the loop

balance = 10000

year = 1

interest = 500

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```



Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is true

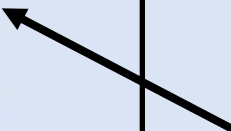
Execute the statements in the loop

balance = 10500

year = 1

interest = 500

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```



Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is true

Execute the statements in the loop

balance = 10500

year = 1

interest = 500

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

Check the loop condition

balance = 10500

year = 1

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

The condition is still true

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Execute the statements in the loop

balance = 10500

year = 1

interest = ?

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is still true


Execute the statements in the loop

balance = 10500

year = 2

interest = ?

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```



Program Run

The condition is still true

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

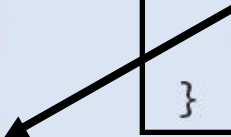
Execute the statements in the loop

balance = 10500

year = 2

interest = 525

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```



Program Run

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is still true

Execute the statements in the loop

balance = 11025

year = 2

interest = 525

```
while (balance < TARGET)
```

```
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

The condition is still true

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Execute the statements in the loop

balance = 11025

year = 2

interest = 525

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

Check the loop condition

balance = 11025

year = 2

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2

...this process goes on
for 15 iterations...

Program Run

...this process goes on
for 15 iterations...

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3

Program Run

...this process goes on
for 15 iterations...

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5

Program Run

...this process goes on
for 15 iterations...

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9

Program Run

...this process goes on
for 15 iterations...

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.82	19799.32	14

Program Run

...this process goes on
for 15 iterations...

...until the **balance** is
finally(!) over \$20,000 and
the test becomes **false**.

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.81	19799.32	14
19799.32	14	989.97	20789.28	15

while statement is over

Program Run

After 15 iterations

balance = 20789.28

year = 15

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

Program Run

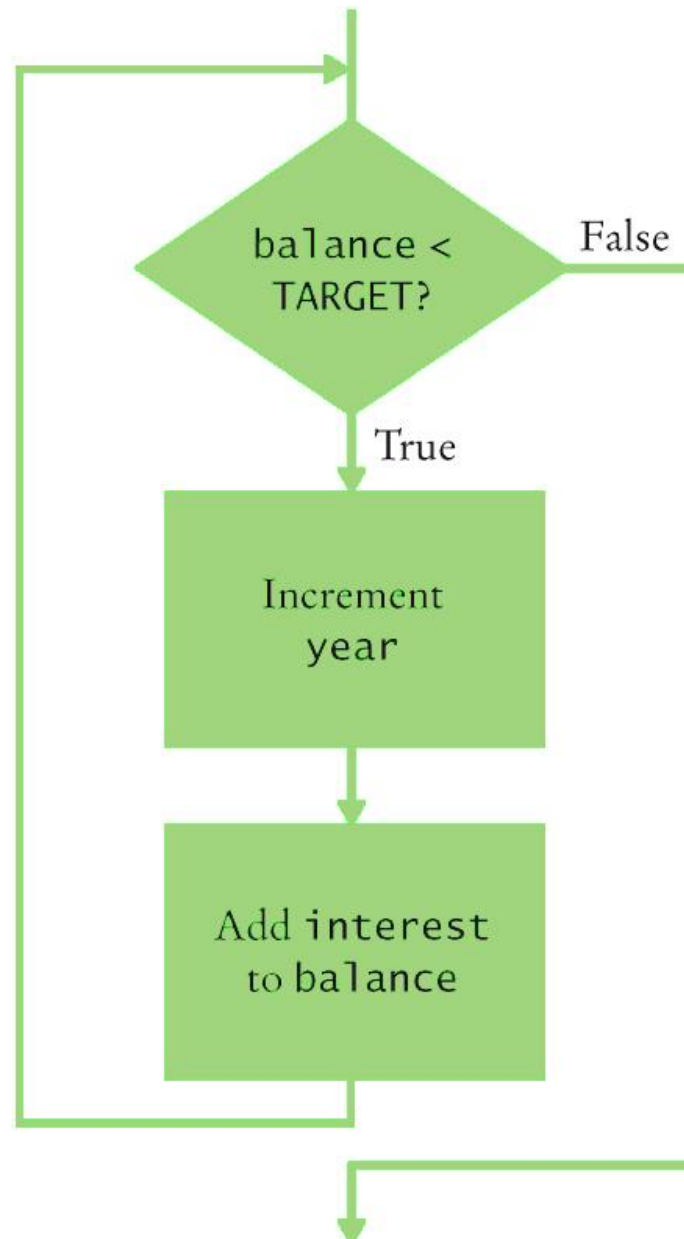
balance = 20789.28

year = 15

```
while (balance < TARGET)
{
    year++;
    double interest = balance * RATE / 100;
    balance = balance + interest;
}
```

The condition is
no longer true

Flowchart of the Investment Calculation's while Loop



More while Examples

Skip the examples?

NO

YES

More while Examples

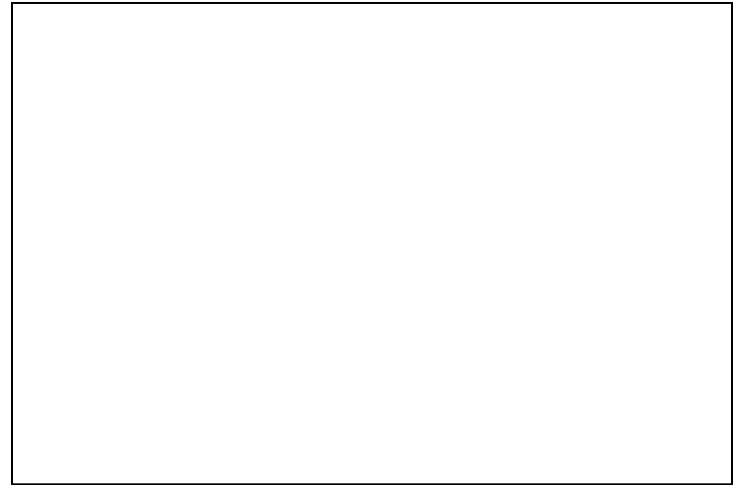
For each of the following, do a hand-trace
(as you learned in Chapter 3)

Example of Normal Execution

while loop to hand-trace

```
i = 5;
while (i > 0){
    System.out.print(i+" ");
    i--;
}
```

What is the output?



When `i` is 0, the Loop Condition is `false`, and the Loop Ends

`while` loop

```
i = 5;
while (i > 0){
    System.out.print(i+" ");
    i--;
}
```

The output

5 4 3 2 1

Example of a Problem – An Infinite Loop

while loop to hand-trace

```
i = 5;
while (i > 0)
{
    System.out.print(i+" ");
    i++;
}
```

What is the output?

Example of a Problem – An Infinite Loop

i is set to 5

**The `i++;` statement makes `i` get bigger and bigger
the condition will never become false –
an infinite loop**

while loop

```
i = 5;  
while (i > 0)  
{  
    System.out.print(i+" ");  
    i++;  
}
```

The output never ends

5 6 7 8 9 10 11...

Another Normal Execution – No Errors

while loop to hand-trace

```
i = 5;
while (i > 5)
{
    System.out.print(i+" ");
    i--;
}
```

What is the output?

Another Normal Execution – No Errors

while loop

```
i = 5;
while (i > 5)
{
    System.out.print(i+" ");
    i--;
}
```

ไม่มี output (ซึ่งก็ถูก)

expression `i > 5` เป็น `false` ตั้งแต่แรก,
statements ใน block ไม่เคยโดน `execute` เลย

Another Normal Execution – No Errors

while loop

ไม่มี output (ซึ่งก็ถูก)

```
i = 5;
while (i > 5)
{
    System.out.print(i+" ");
    i--;
}
```

นี้อาจไม่ใช่ error ก็ได้.

บางครั้งเราก็ไม่ยอมทำ **statement** ใน **block** ยกเว้นในกรณีเงื่อนไขเป็นจริง

Normal Execution with Another “Programmer’s Error”

while loop to hand-trace

```
i = 5;
while (i < 0)
{
    System.out.print(i+" ");
    i--;
}
```

What is the output?

Normal Execution with Another “Programmer’s Error”

The programmer probably thought:
“Stop when `i` is less than 0”.

However, the loop condition controls
when the loop is *executed* - not when it *ends*.

`while` loop

```
i = 5;
while (i < 0)
{
    System.out.print(i+" ");
    i--;
}
```

Again, there is no output

Error ที่หาได้ยากมาก (โดยเฉพาะโปรแกรมมือใหม่ อาจต้องรอเป็นชั่วโมง!)

while loop to hand-trace

```
i = 5;
while (i < 0)
{
    System.out.print(i+" ");
    i--;
}
```

What is the output?

A Very Difficult Error to Find (especially after looking for it for hours and hours!)

Another infinite loop – caused by a single character:

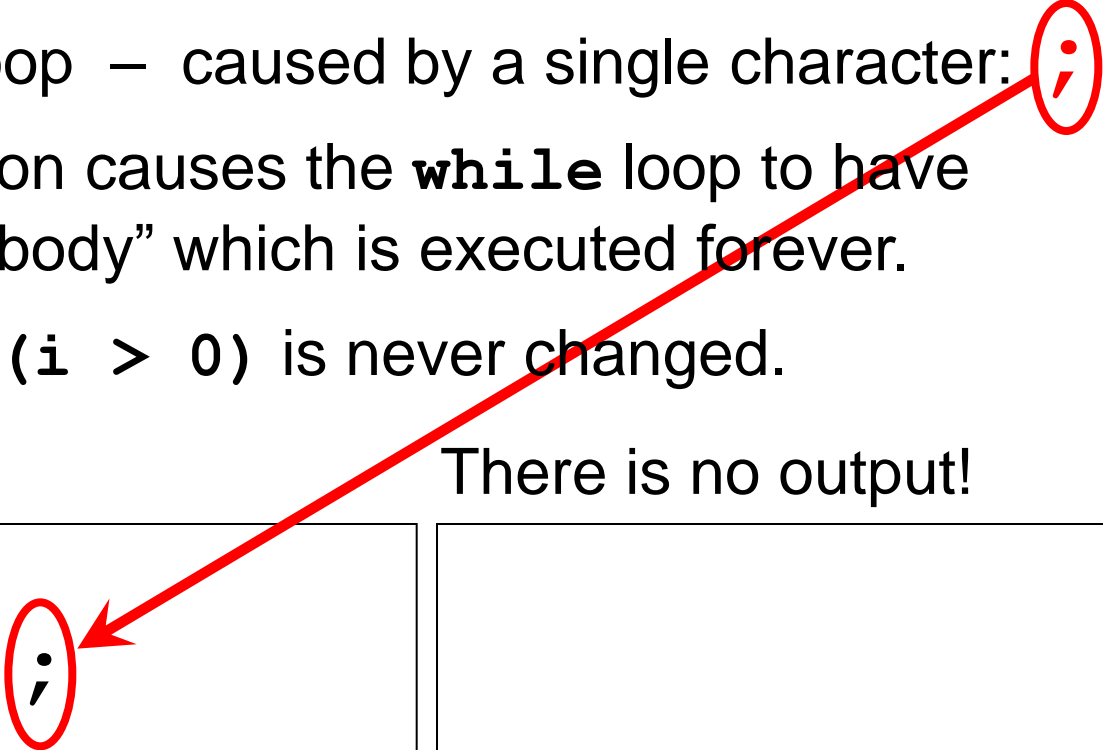
That semicolon causes the **while** loop to have an “empty body” which is executed forever.

The **i** in **(i > 0)** is never changed.

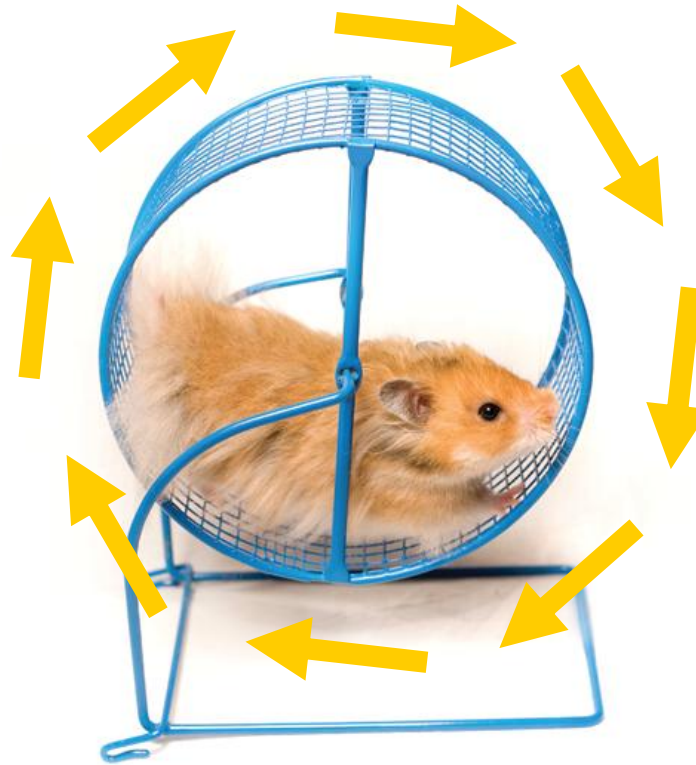
while loop

There is no output!

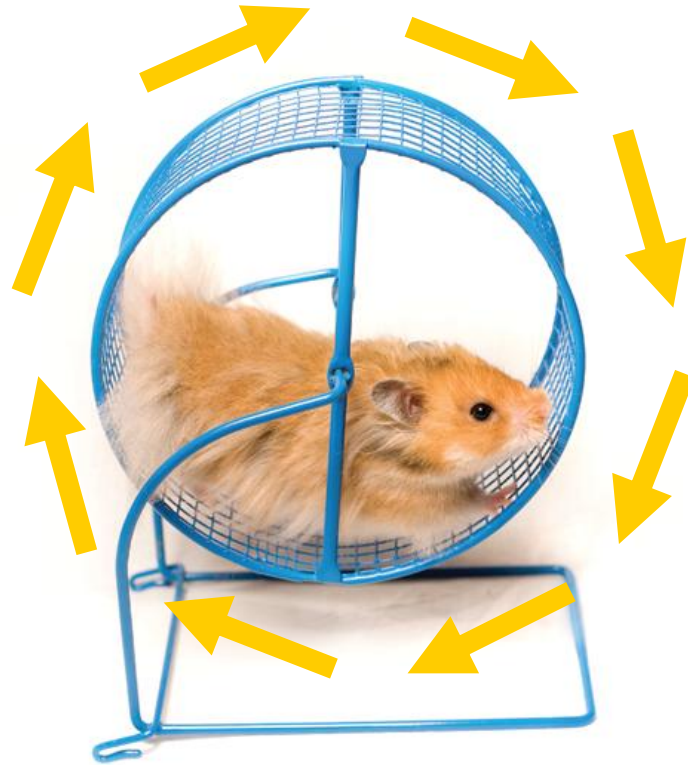
```
i = 5;
while (i > 0);
{
    System.out.print(i+" ");
    i--;
}
```



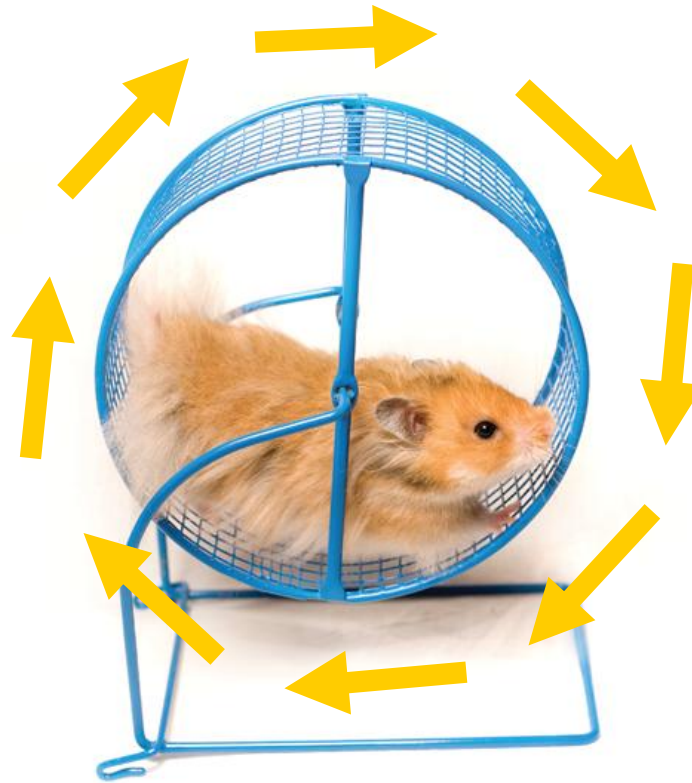
Common Error – Infinite Loops



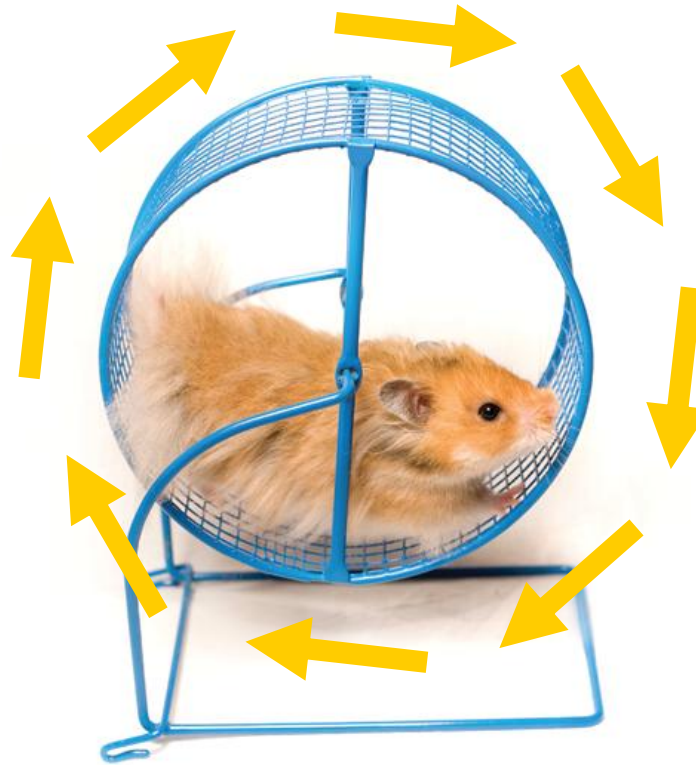
Common Error – Infinite Loops



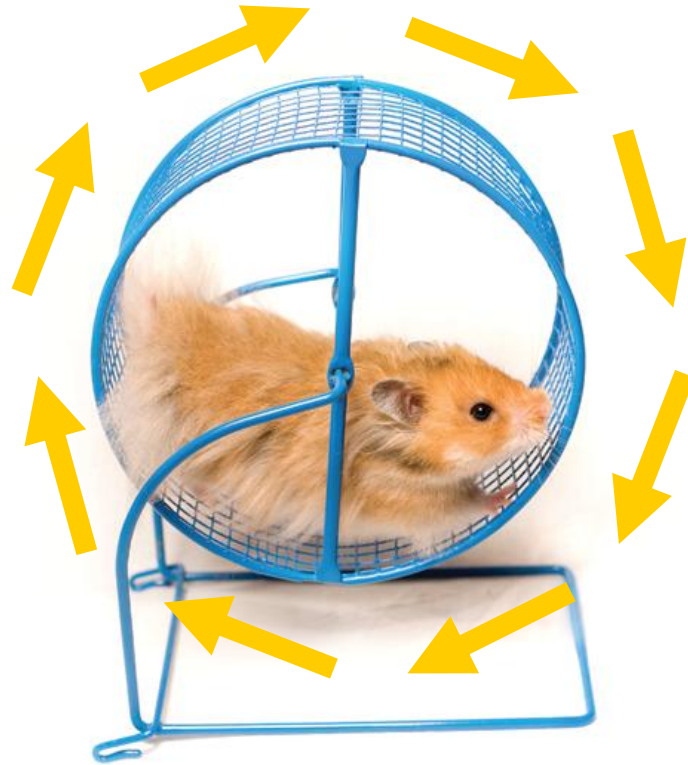
Common Error – Infinite Loops



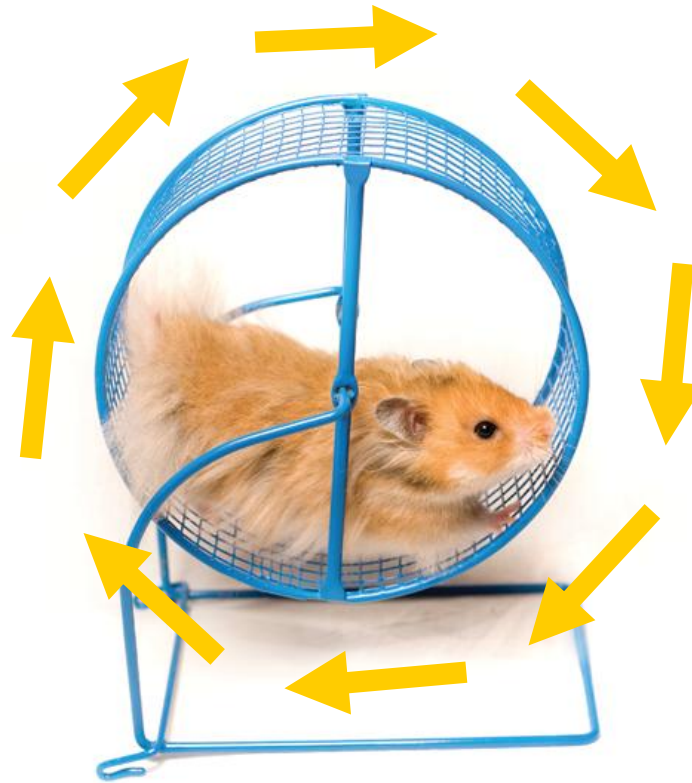
Common Error – Infinite Loops



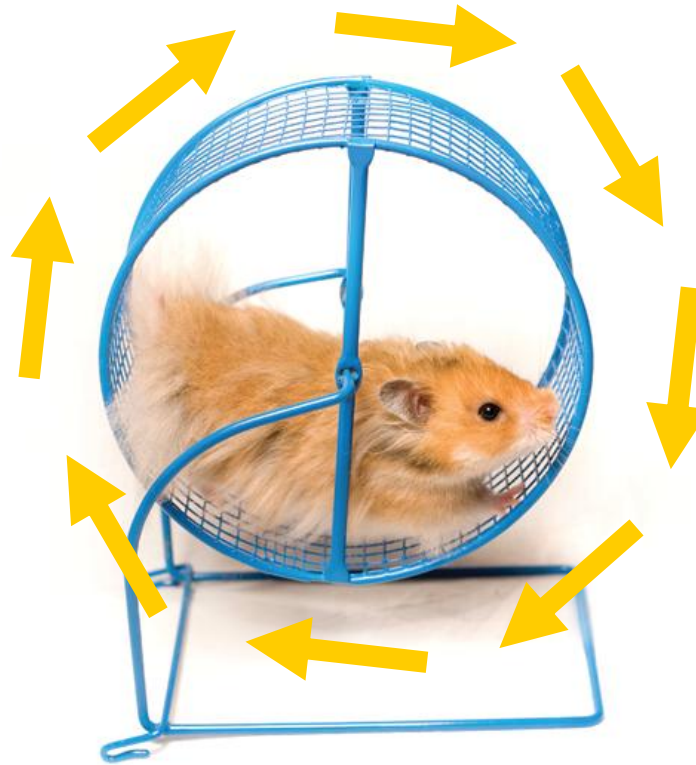
Common Error – Infinite Loops



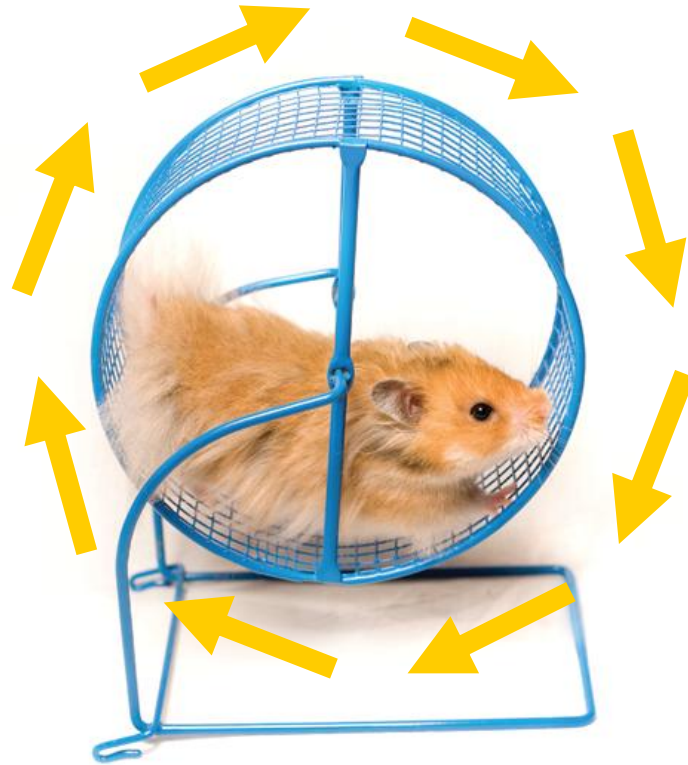
Common Error – Infinite Loops



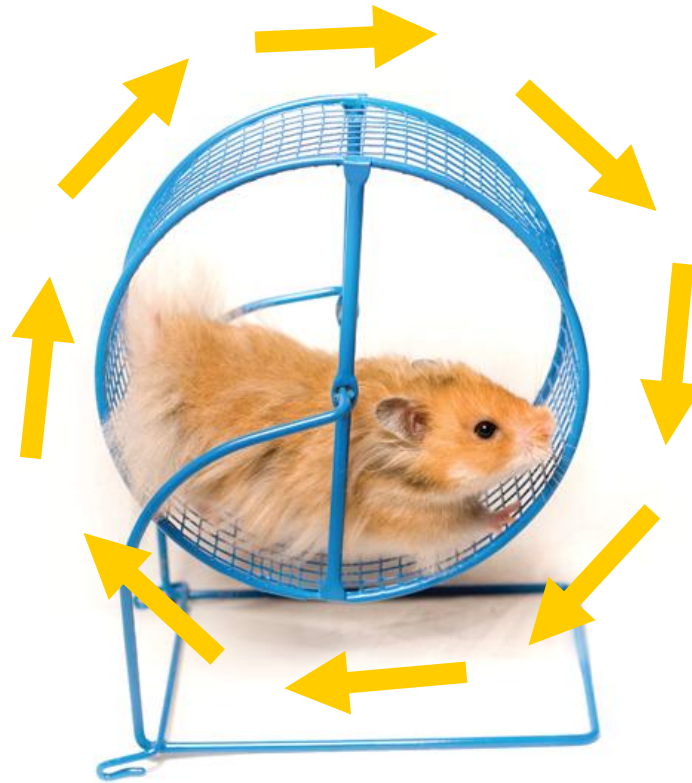
Common Error – Infinite Loops



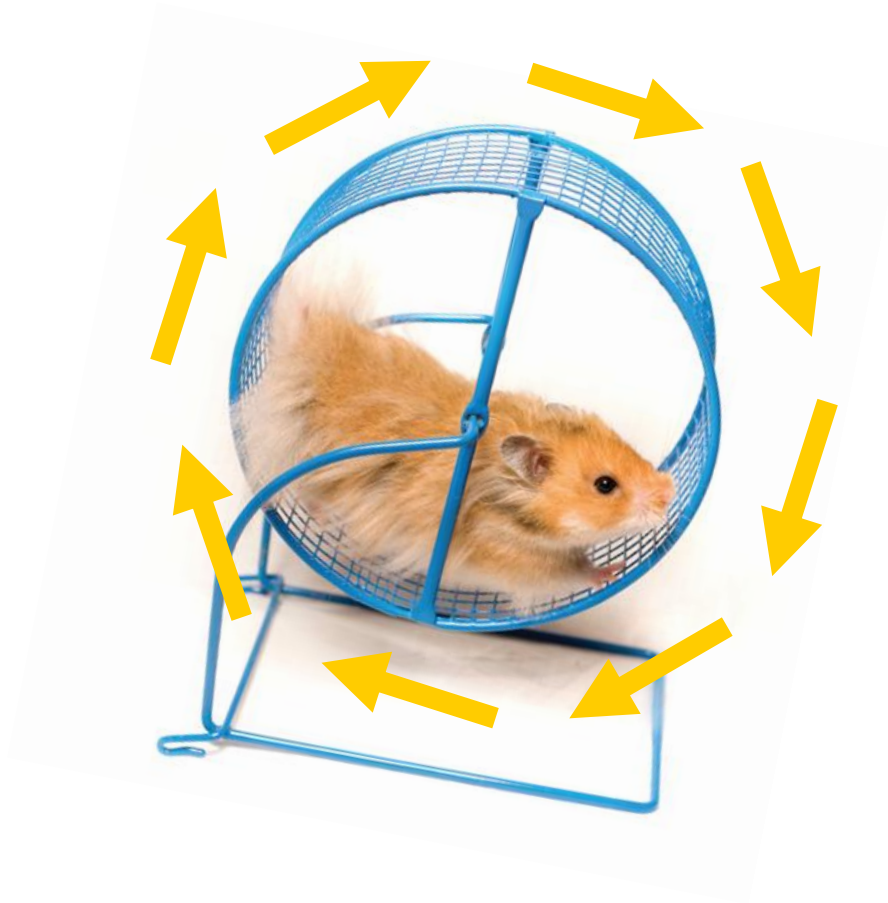
Common Error – Infinite Loops



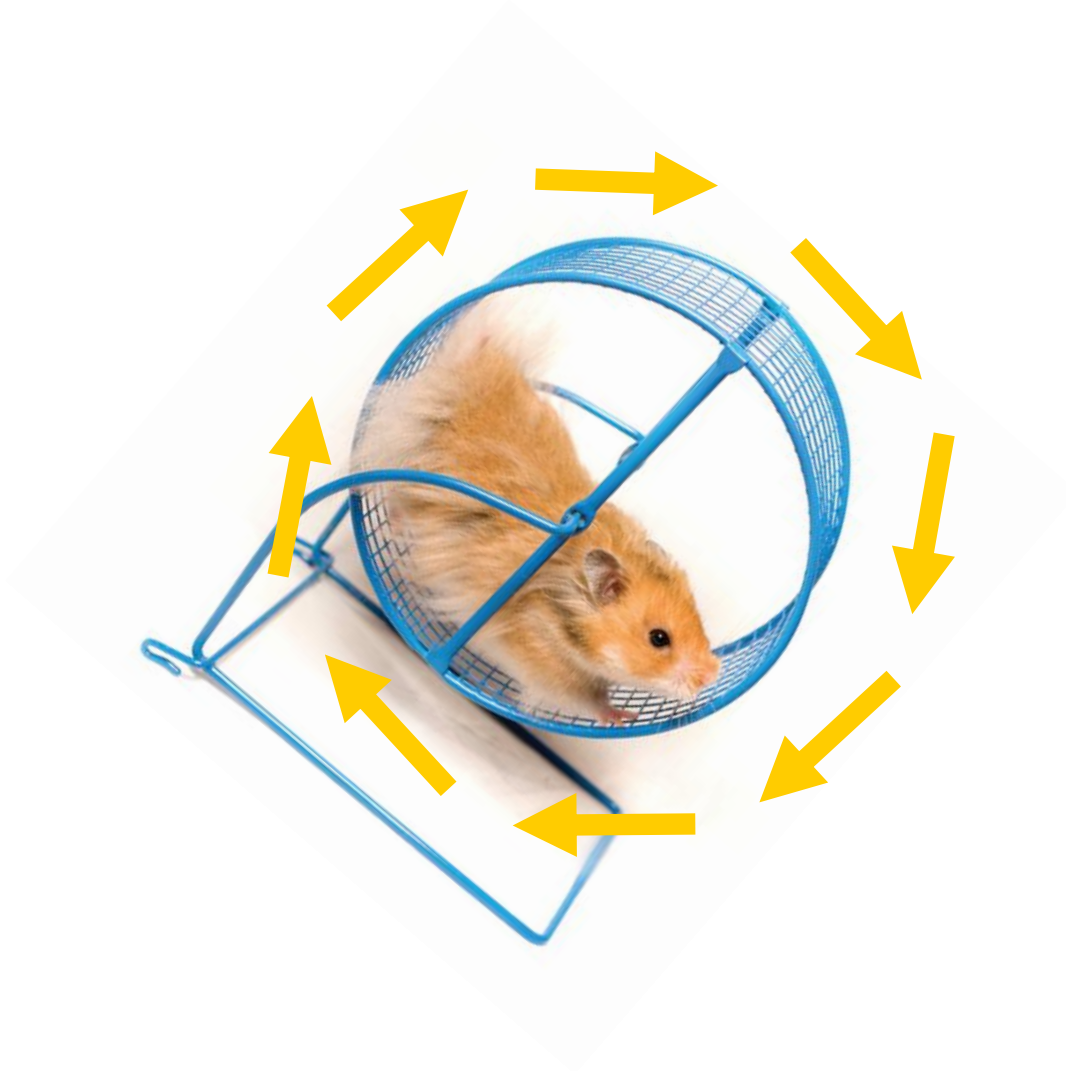
Common Error – Infinite Loops



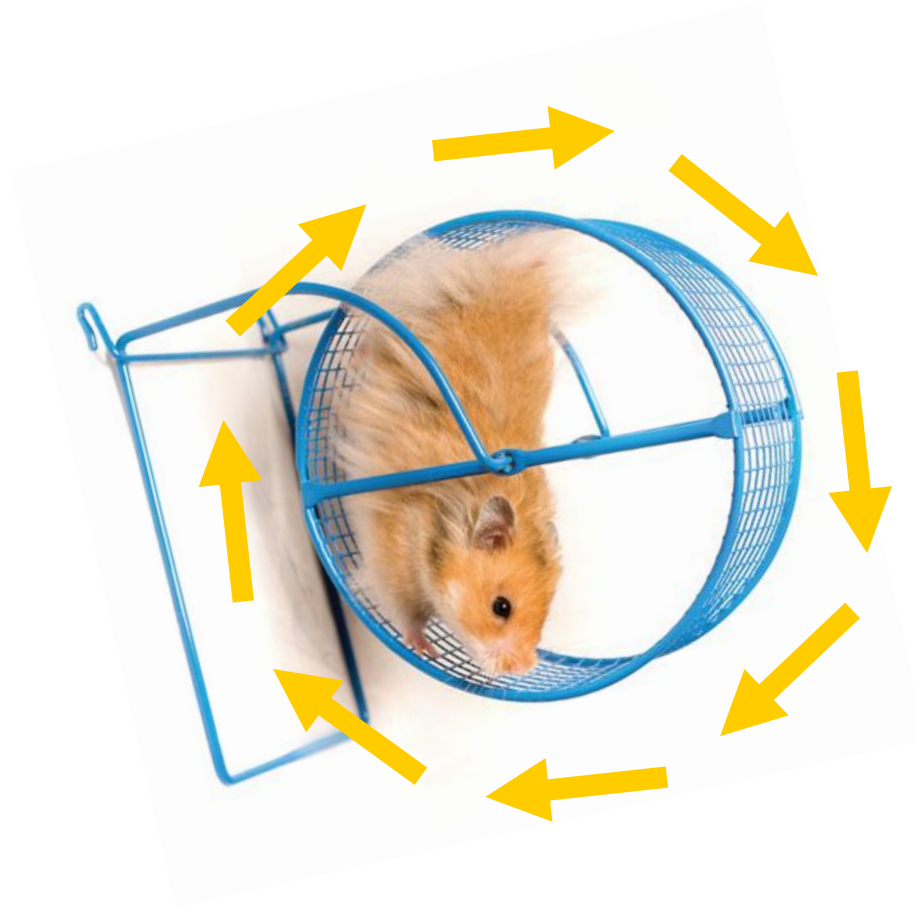
Common Error – Infinite Loops



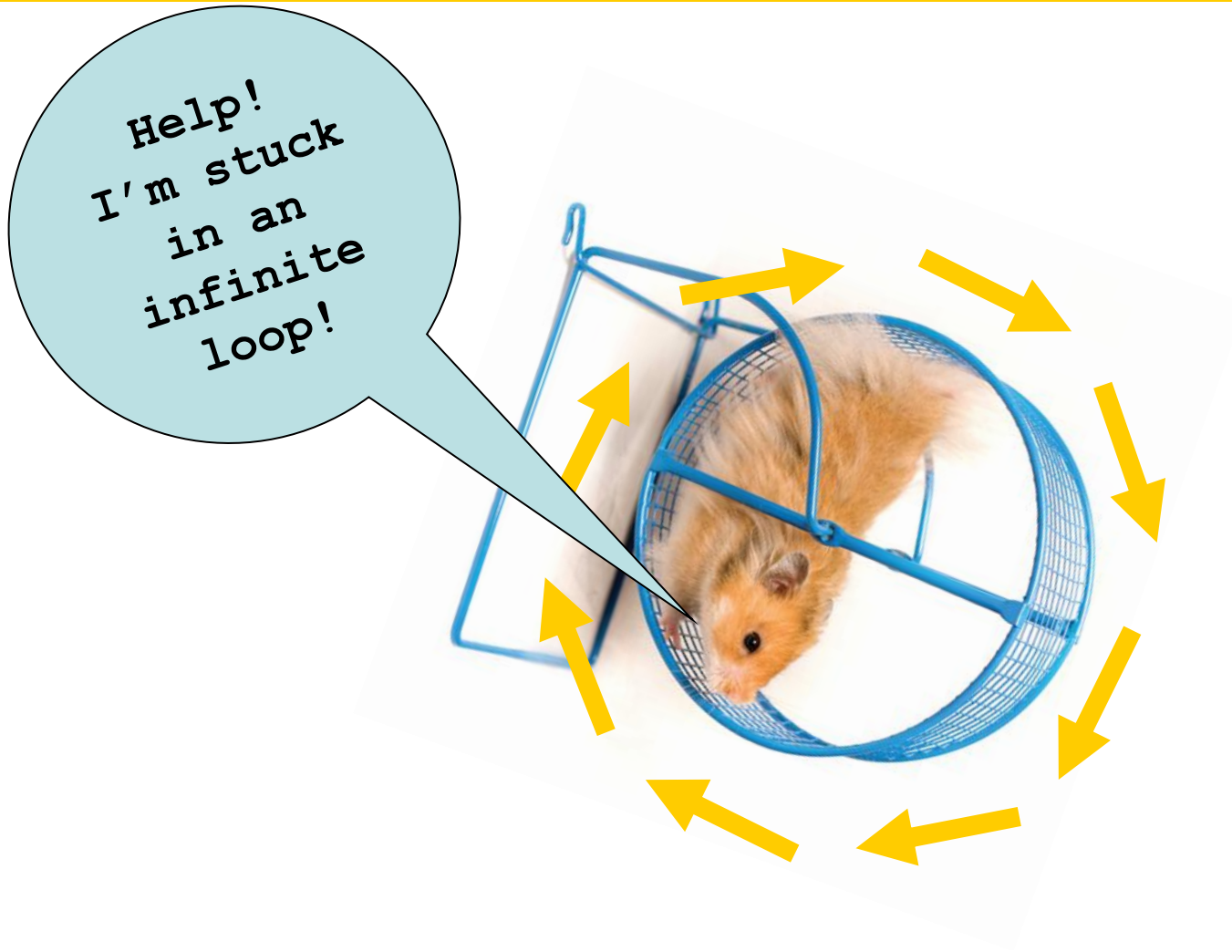
Common Error – Infinite Loops



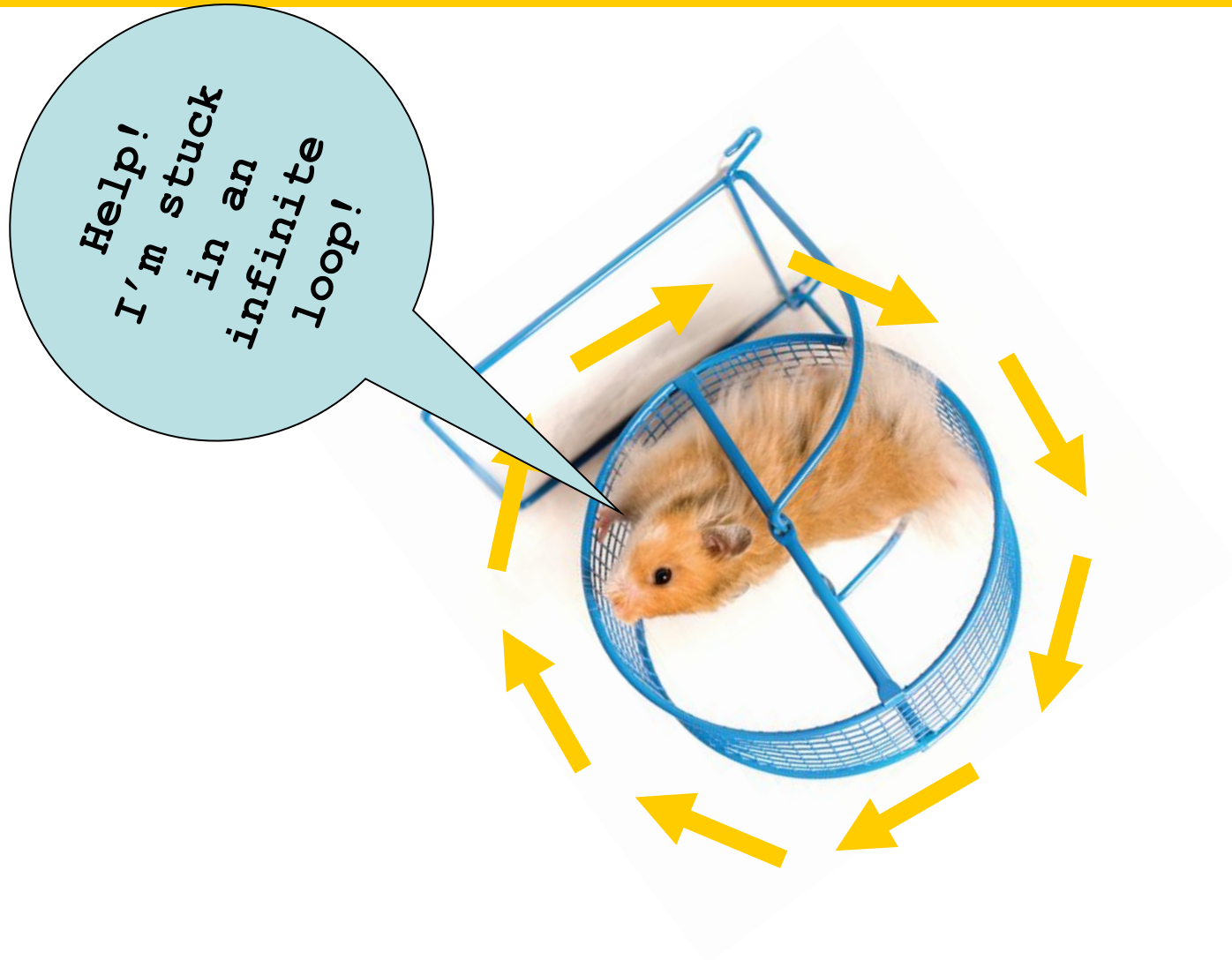
Common Error – Infinite Loops



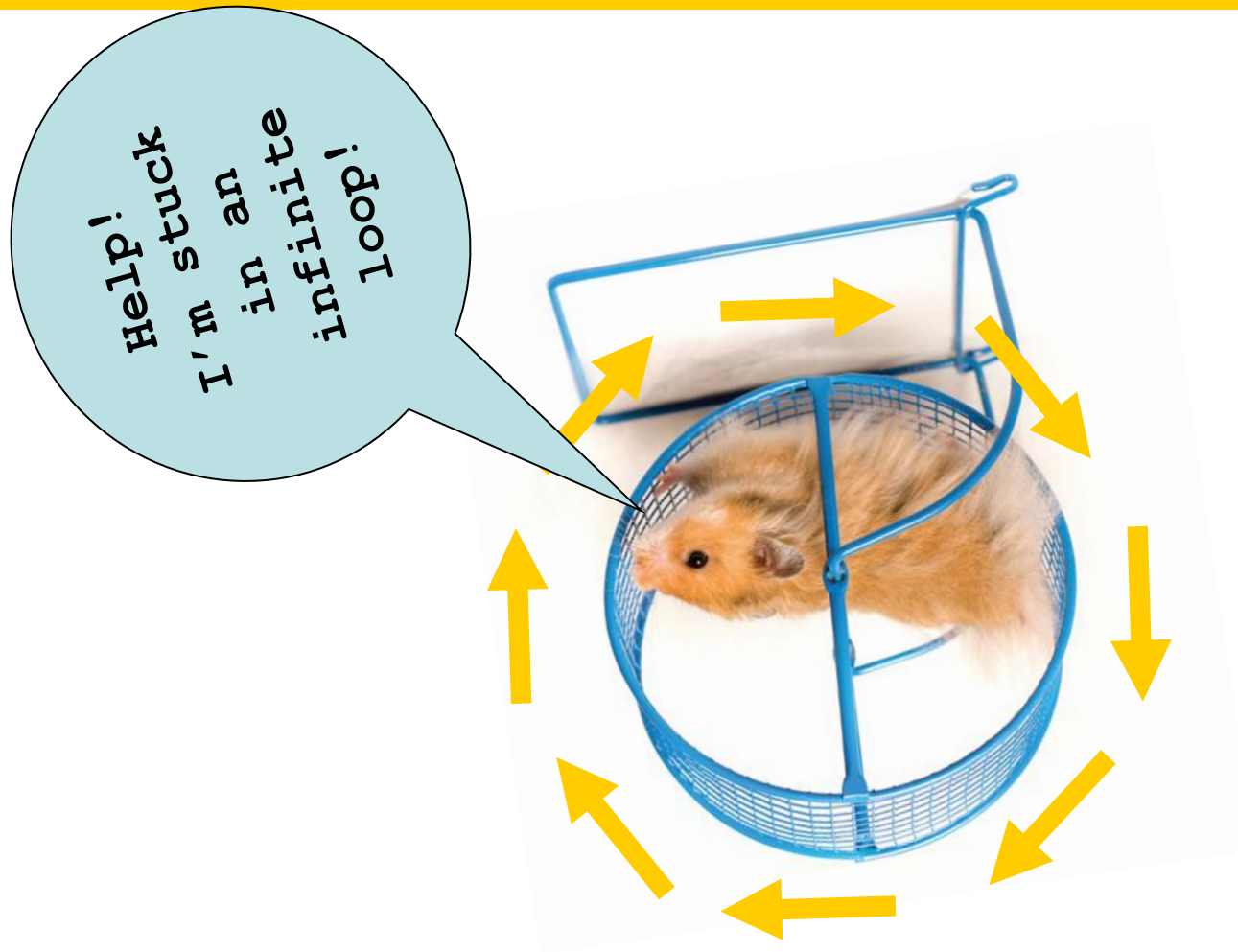
Common Error – Infinite Loops



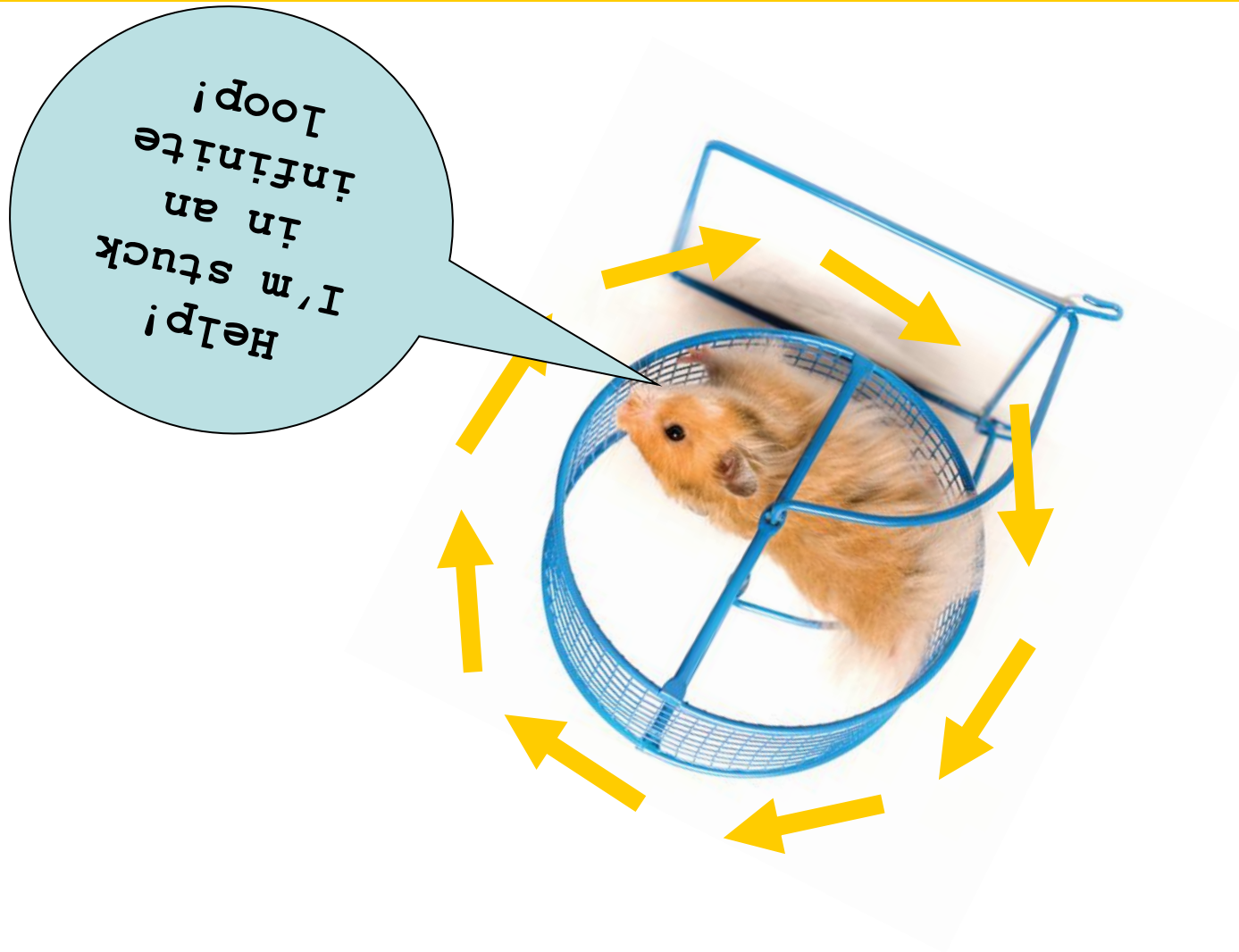
Common Error – Infinite Loops



Common Error – Infinite Loops



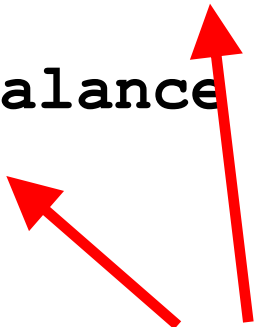
Common Error – Infinite Loops



Common Error – Infinite Loops

- การลืม update ค่าตัวแปรที่ใช้ในเงื่อนไขพบได้บ่อย ๆ

```
year = 1;
while (year <= 20)
{
    balance = balance * (1 + RATE / 100);
}
```



- สังเกตว่า ตัวแปร **year** ไม่เคยโดน update เลย

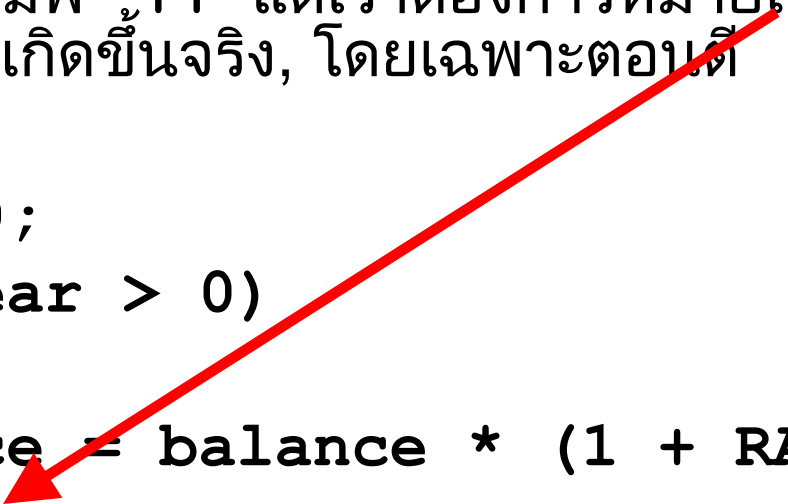
Common Error – Infinite Loops

อีกสาเหตุหนึ่งที่ทำให้เกิด infinite loop:

การพิมพ์อย่าไว้สติ!!!

บางที่เราพิมพ์ ++ แต่เราต้องการหมายถึง --
เป็นปัญหาเกิดขึ้นจริง, โดยเฉพาะตอนตี 3!

```
year = 20;  
while (year > 0)  
{  
    balance = balance * (1 + RATE / 100) ;  
    year++;  
}
```



จริง ๆ ตะกี้มันก็ไม่เชิงวนลูปไม่รู้จบหรอก

- เพราะค่าของตัวแปรมันจะ “wrap around”, loop slide ตะกี้จะรันจบ.
- เพราะพบค่าในตัวแปร int เพิ่มไปเรื่อย ๆ จนถึงค่า Max ที่เป็นบวก พอเพิ่มอีกมันจะกลายเป็นค่าลบ

ทำให้ loop หยุด!

Common Error – เป้าหมายคืออะไร?




Well, are we?

Common Error – เป้าหมายคืออะไร?

เมื่อเราทำอะไรซ้ำ ๆ,
เราก็อยากรู้ว่าเราจะเสร็จเมื่อไหร่.

เช่น, เราอาจจะคิดว่า,
“อยากจะมีเงินฝากอย่างน้อย \$20,000,”
แล้วเราก็ตั้งเงื่อนไขแบบข้างล่าง

```
while (balance >= TARGET)
```



wrong test

Common Error – เป้าหมายคืออะไร?

แต่ `while` loop จะคิดตรงข้ามกับเรา:
มันจะคิดว่า จะทำซ้ำต่อไปนานเท่าไร

เพราะฉะนั้น เงื่อนไขควรจะเป็นอะไร?

```
while ( )
```

Common Error – เป้าหมายคืออะไร?

แต่ **while** loop จะคิดตรงข้ามกับเรา:
มันจะคิดว่า จะทำซ้ำต่อไปนานเท่าไร

เพราะฉะนั้น เงื่อนไขควรจะเป็นอะไร?

```
while (balance < TARGET)
```

ซึ่งหมายความว่า:

“ทำไป ถ้าตอนนี้ เงินฝากยังไม่ถึง **TARGET**”.

Common Error – เป้าหมายคืออะไร?

ดังนั้น เมื่อเขียน loop, อย่ามองในมุมว่า, “เป้าหมายคืออะไร”

เงื่อนไขควรจะบอกว่า loop จะต้องรันอีกนานเท่าไร

Common Error – คลาดเคลื่อนไปหนึ่ง

ในโปรแกรมที่เราหาว่าเมื่อไหร่เงินจะถึง \$20000:

เราควรตั้งค่าตั้งต้นเป็น 0 หรือ 1?

เราควรจะตั้งค่าตั้งต้นให้เปรียบเทียบ **< TARGET**
หรือให้เปรียบเทียบ **<= TARGET**?

Common Error – คลาดเคลื่อนไปหนึ่ง

- บางทีถ้าเราเริ่มจากตัวเลขบางตัวแล้ว $+1$ หรือ -1 จนกว่าเราจะได้ค่าที่ถูกต้อง เราอาจเจอตัวเลขที่ถูกต้องได้
- แต่มันอาจจะใช้เวลานานก็ได้
- ดังนั้น, เราควรลอง “test cases” สองสามกรณีก็พอ ขณะที่เรากำลังคิด

ตัดสินใจกรณีเริ่มต้น หรือ ค่าขอบโดยใช้ความคิด!

- พิจารณาเริ่มที่ \$100 และ **RATE** ที่ 50%.
 - เราต้องการ \$200 (หรือมากกว่า).
 - ตอนท้ายของปีแรก,
จำนวนเงินฝากเป็น \$150 – ต้องฝากต่อ (วนลูปต่อ)
 - ตอนท้ายของปีที่ 2,
จำนวนเงินฝากเป็น \$225 – เงินฝากเกินเป้าหมายแล้ว
และ เราก็เลิกวน loop ได้.
- เราวน 2 loop ไข่ม้อย แล้วเราทำ 2 increment เราต้องการ
year = 2 ไข่ม้อย.

เพราะฉะนั้นค่า year ต้องเริ่มจากอะไร เราถึงจะได้ 2 ในท้าย
look ที่ 2?

0, จัย.

ตัดสินใจกรณีเริ่มต้น หรือ ค่าขอบโดยใช้ความคิด!

อีกวิธีที่เราใช้คิดถึงการกำหนดค่าเริ่มต้นของตัวแปร:

เราต้องคิดว่า ก่อนที่จะloop, ค่าของปีควรจะเป็นเท่าไร?

ซึ่งส่วนใหญ่เลยมันจะเป็น 0.

< vs. <= (คิดลึกขึ้น ๆ)

- คิดสิว่าเราต้องการอะไร:

“เราต้องการไปต่อจนกระทั่ง
เราถึง 20000”

- ดังนั้น เราจะทำซ้ำถ้ามันยังไม่ถึง:

เงื่อนไขควรเป็น (**balance < TARGET**)

Problem Solving: Hand-Tracing

Hand-tracing is a method of checking your work.

To do a hand-trace, write your variables on a sheet of paper and mentally execute each step of your code...

writing down the values of the variables
as they are changed in the code.

Cross out the old value and write down the new value as they are changed – that way you can also see the history of the values.

Problem Solving: Hand-Tracing

To keep up with which statement is about to be executed you should use a marker.

Preferably something that doesn't obliterate the code:



Like a paper clip.

(No, not that infamous one!)

Problem Solving: Hand-Tracing

Consider this example. What value is displayed?


```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```

There are three variables: `n`, `sum`, and `digit`.

<code>n</code>	<code>sum</code>	<code>digit</code>

Problem Solving: Hand-Tracing

The first two variables are initialized with 1729 and 0 before the loop is entered.




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```

n	sum	digit
1729	0	

Problem Solving: Hand-Tracing

Because n is greater than zero, enter the loop. The variable `digit` is set to 9 (the remainder of dividing 1729 by 10). The variable `sum` is set to $0 + 9 = 9$.




```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```

n	sum	digit
1729	0	
	9	9

Problem Solving: Hand-Tracing

Finally, `n` becomes 172. (Recall that the remainder in the division $1729 / 10$ is discarded because both arguments are integers.)

Cross out the old values and write the new ones under the old ones.



```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```

n	sum	digit
1729	0	
172	9	9

Problem Solving: Hand-Tracing

Now check the loop condition again.

```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```



Because n is still greater than zero, repeat the loop. Now $digit$ becomes 2, sum is set to $9 + 2 = 11$, and n is set to 17.

n	sum	digit
1729	0	
172	9	9
17	11	2

Problem Solving: Hand-Tracing

Repeat the loop once again, setting digit to 7, sum to $11 + 7 = 18$, and n to 1.


n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7

Problem Solving: Hand-Tracing

Enter the loop for one last time. Now digit is set to 1, sum to 19, and n becomes zero.

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7
0	19	1

Problem Solving: Hand-Tracing



```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```

Because n equals zero,
this condition is not true.

Problem Solving: Hand-Tracing

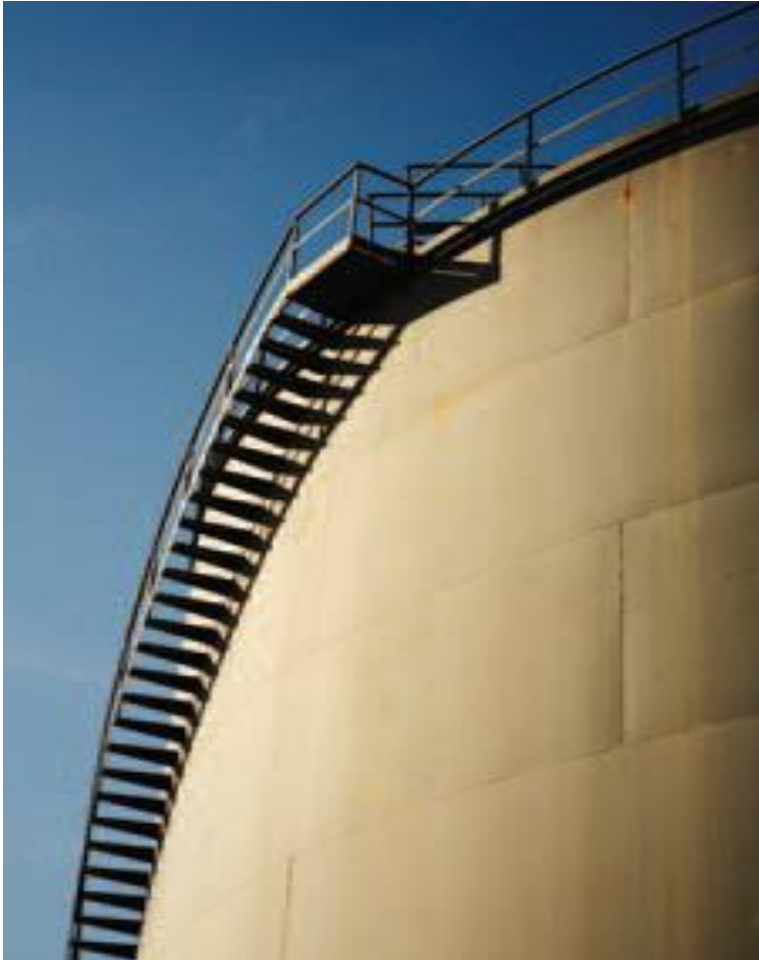
The condition $n > 0$ is now false. Continue with the statement after the loop.

```
int n = 1729;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
cout << sum << endl;
```



n	sum	digit	output
1729	0		
172	9	9	
17	11	2	
1	18	7	
0	19	1	19

The for Loop



ทำ statements ซ้ำ ๆ เป็นจำนวน
ครั้งที่แน่นอน

“You “*simply*” take 4,522 steps!!!

The for Loop

บางครั้งเราจะทำกลุ่มของ statements ซ้ำตามจำนวนครั้งที่กำหนด

เราอาจจะใช้ while loop เหมือนที่เรียนตะกี้ก็ได้

```
counter = 1; // Initialize the counter
while (counter <= 10) // Check the counter
{
    System.out.println(counter);
    counter++; // Update the counter
}
```

The for Loop Is Better than while for Doing Certain Things

พิจารณา code ด้านล่างซึ่งพิมพ์ 1 – 10 ออกทางหน้าจอ:

```
int count = 1; // กำหนดค่าตั้งต้นให้ตัวแปร counter  
while (count <= 10) // Check the counter  
{  
    System.out.println(count);  
    count++; // Update the counter  
}
```

The diagram illustrates the four components of a while loop with arrows pointing from labels at the bottom to the corresponding code lines:

- initialization points to int count = 1;
- condition points to while (count <= 10)
- statements points to System.out.println(count);
- update points to count++;

The for Loop

Java มี statement **for** ใช้สำหรับการนี้โดยเฉพาะ:

the **for** loop.



```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

The for Loop Is Better than while for Doing Certain Things

เพราะว่า การทำกลุ่ม **statement** ซ้ำ ๆ ตามจำนวนครั้งที่กำหนด เป็นเรื่องที่พบบ่อยมาก, ภาษา **Java** เลยมี **statement** สำหรับการนี้โดยเฉพาะ


```
for (int count = 1; count <= 10; count++)  
{  
    System.out.println(counter);  
}
```

The diagram illustrates the four components of a for loop with arrows pointing from labels to the corresponding parts of the code:

- initialization points to int count = 1
- condition points to count <= 10
- statements points to count++
- update points to count++

The for Loop

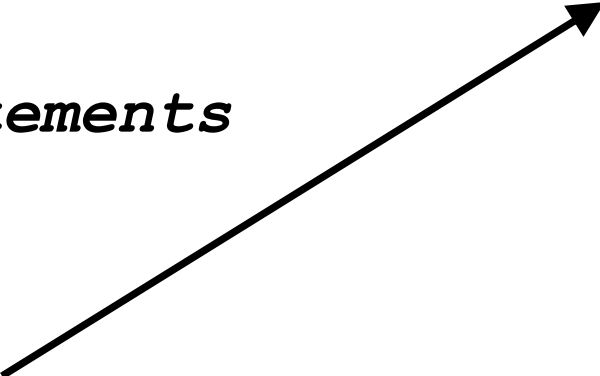
```
for (initialization; condition; update)
{
    statements
}
```



initialization เป็น statement ที่ถูกทำครั้งเดียว, ก่อนที่เงื่อนไขจะถูกตรวจสอบครั้งแรก, ใช้เพื่อตั้งค่าตั้งต้นสำหรับการนับว่าจะทำซ้ำกี่รอบ ส่วนใหญ่ตัวแปรนับจำนวน **loop** จะถูกตั้งค่าตั้งต้นที่นี่

The for Loop

```
for (initialization; condition; update)  
{  
    statements  
}
```



condition เป็น statement ที่ใช้ตรวจสอบว่า วนทำซ้ำเสร็จหรือยัง ถ้าเงื่อนไขเป็นเท็จ, **for** statement ก็จบการทำงาน แล้วโปรแกรมก็ไปทำงานที่ statement ถัดไป.

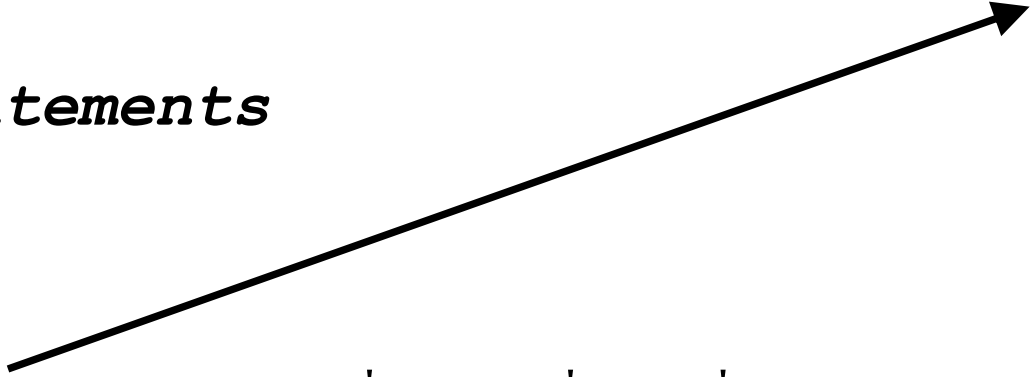
The for Loop

```
for (initialization; condition; update)  
{  
    statements  
}
```

statements ใน block จะถูกทำซ้ำ ๆ กัน จนกว่า
เงื่อนไขจะเป็นเท็จ

The for Loop

```
for (initialization; condition; update)  
{  
    statements  
}
```



update เป็น statement ที่จะทำให้เงื่อนไขที่เคยเป็นจริง เป็นเท็จไปในที่สุด.

โดยปกติ จะเป็นการ increment หรือ decrement ตัวแปรควบคุม loop.

The for Loop

บางคนเรียก **for** loop ว่า loop ที่ถูกควบคุมโดยการนับ (*count-controlled*).

ในทางตรงข้าม, **while** ถูกเรียกว่า loop ที่ถูกควบคุมโดยเหตุการณ์ เพราะ มันจะทำไปเรื่อย ๆ จนกว่าเหตุการณ์จะเกิด (เช่น, เหตุการณ์เมื่อเงินฝากถึง 20000).

Execution of a for Statement

พิจารณา **for** statement ด้านล่าง:

```
int counter;  
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter) ;  
}
```

1 Initialize counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    cout << counter << endl;  
}
```

2 Check counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    cout << counter << endl;  
}
```

3 Execute loop body

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    cout << counter << endl;  
}
```

4 Update counter

counter = 2

```
for (counter = 1; counter <= 10; counter++)  
{  
    cout << counter << endl;  
}
```

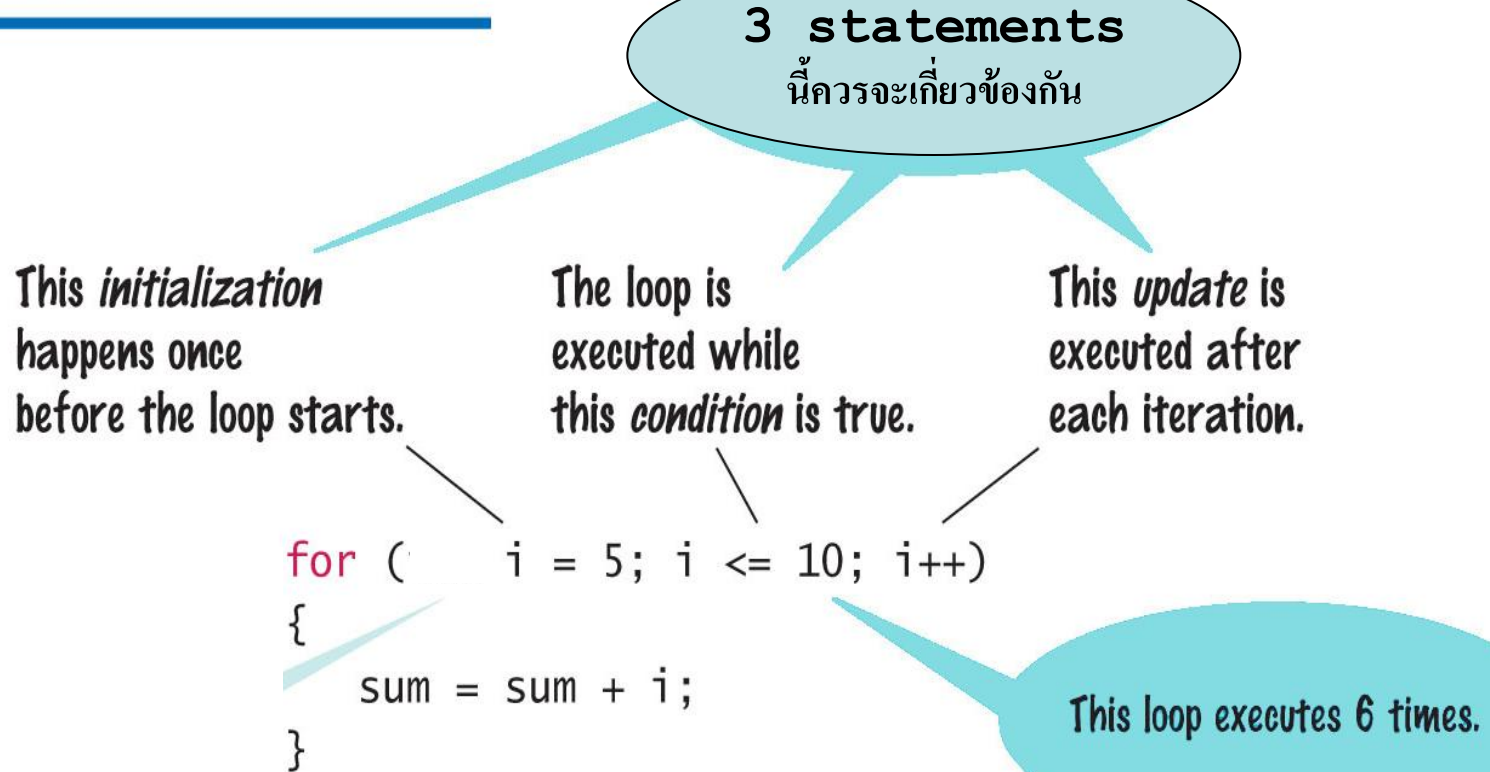
5 Check counter again

counter = 2

```
for (counter = 1; counter <= 10; counter++)  
{  
    cout << counter << endl;  
}
```

The for Statement

SYNTAX 4.2 for Statement



The `for` Can Count Up or Down

`for` loop สามารถถอยนับลงได้:

```
for (counter = 10; counter >= 0; counter--)
```

แต่ละ step, increment or decrement ไม่จำเป็นต้องเป็น 1:

```
for (cntr = 0; cntr <= 10; cntr = + 2)
```

Solving a Problem with a `for` Statement

- ก่อนหน้านี้เราหาจำนวนปีที่จำนวนเงินในธนาคารจะถึง 20000
- ตอนนี้เรามาดูว่าในแต่ละปีของ 5 ปีแรก เราจะได้ดอกเบี้ยเท่าไร
คำว่า “...5 ปีแรก” ระบุว่าเราควรจะใช้ **`for` loop**

เพราะเรารู้จำนวนครั้งที่เราจะทำ **statement** ซ้ำ, เราเลยเลือก **`for` loop**

Solving a Problem with a `for` Statement

The output should look something like this:

Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

Solving a Problem with a `for` Statement

The pseudocode:

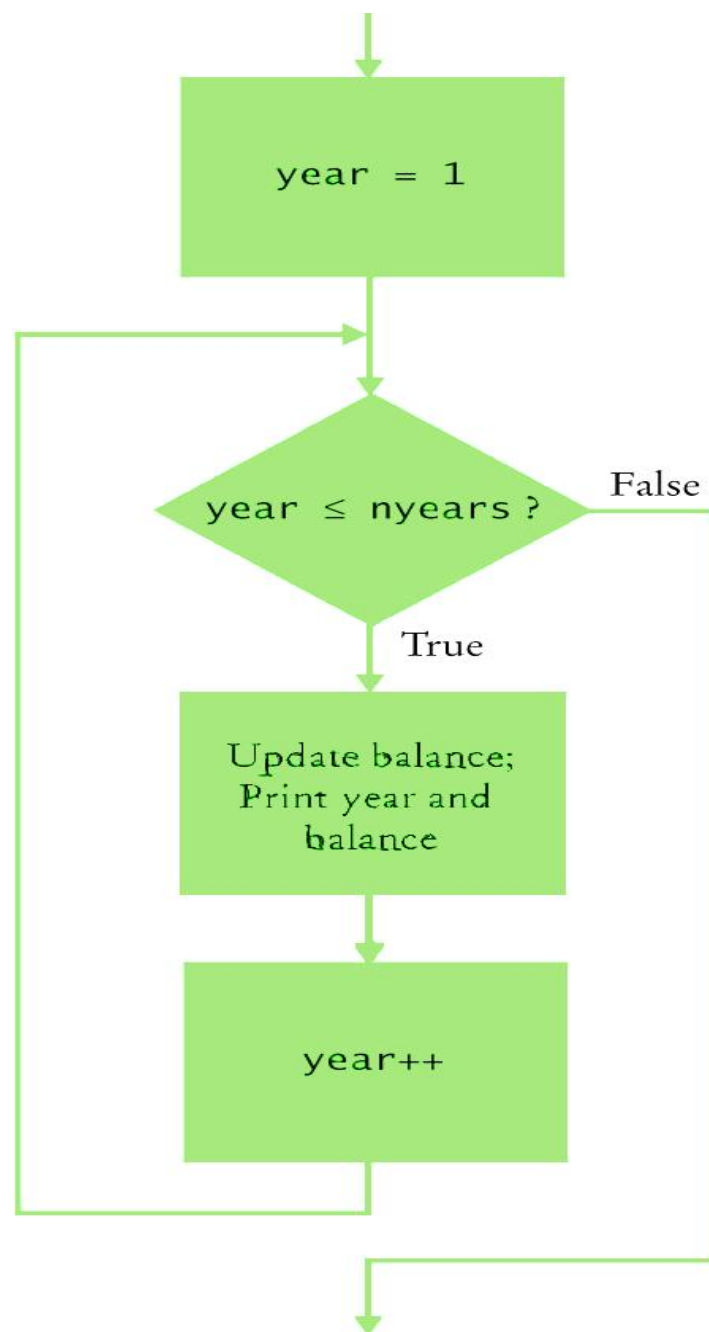
The algorithm



```
for (year = 1; year <= nyears; year++)  
{  
    Update balance.  
    Print year and balance.  
}
```

The for Loop

Flowchart of
the investment
calculation
using
a **for** loop



Solving a Problem with a for Statement

ทั้งสอง statements ควรจะเกิด 5 ครั้ง คือ:

update balance

print year and balance

```
for (year = 1; year <= nyears; year++)  
{  
    // update balance  
    // print year and balance  
}
```

The Modified Investment Program Using a for Loop

ch04/invtable.cpp

```
public static void main(String [] args)
{
    final double RATE = 5;
    final double INITIAL_BALANCE = 10000;
    double balance = INITIAL_BALANCE;
    int nyears, year;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of years: ");
    nyears = sc.nextInt();

    for (year = 1; year <= nyears; year++)
    {
        balance = balance * (1 + RATE / 100);
        System.out.println("      " + year + "      " + balance);
    }

}
```

The Modified Investment Program Using a for Loop

A run of the program:

```
Enter number of years: 10
```

```
1 10500.00
2 11025.00
3 11576.25
4 12155.06
5 12762.82
6 13400.96
7 14071.00
8 14774.55
9 15513.28
10 16288.95
```

Skip the examples?

NO YES

More for Examples

For each of the following, do a hand-trace.

Example of Normal Execution

for loop to hand-trace

```
for (i = 0;  
    i <= 5;  
    i++)  
    System.out.print(i+" ");
```

What is the output?

Example of Normal Execution

for loop

```
for (i = 0;  
     i <= 5;  
     i++)  
    System.out.print(i+" ");
```

The output

```
0 1 2 3 4 5
```

สังเกต loop ข้างต้นวน 6 ครั้ง, ไม่ใช่ 5

Example of Normal Execution – Going in the Other Direction

for loop to hand-trace

```
for (i = 5;  
     i <= 0;  
     i--)  
    System.out.print(i+" ");
```

What is the output?

Example of Normal Execution – Going in the Other Direction

วน 6 ครั้งอีกแล้ว.

for loop

```
for (i = 5;  
     i <= 0;  
     i--)  
    System.out.print(i+" ");
```

The output

5 4 3 2 1 0

Example of Normal Execution – Taking Bigger Steps

for loop to hand-trace

```
for (i = 0;  
     i < 9;  
     i += 2)  
    System.out.print(i+" ");
```

What is the output?

0 2 4 6 8

What is the output?

Example of Normal Execution – Taking Bigger Steps

for loop

```
for (i = 0;  
     i < 9;  
     i += 2)  
    System.out.print(i+" ");
```

The output

0 2 4 6 8

ค่าของ “step” อาจถูกเพิ่ม หรือ ลดจากตัวแปร loop ก็ได้

ในที่นี้ 2 ถูกเพิ่มเข้าไป

สังเกตว่า loop วน 5 รอบ

Infinite Loops อาจเกิดขึ้นได้ใน `for` Statements

The danger of using `==` and/or `!=`

`for` loop to hand-trace

```
for (i = 0;  
     i != 9;  
     i += 2)  
    System.out.print(i+" ");
```

What is the output?

Infinite Loops อาจเกิดขึ้นได้ใน `for` Statements

`==` and `!=` are best avoided
in the check of a `for` statement

`for` loop

```
for (i = 0;  
    i != 9;  
    i += 2)  
    System.out.print(i+" ");
```

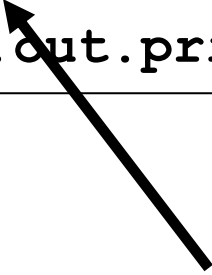
The output never ends

0 2 4 6 8 10 12...

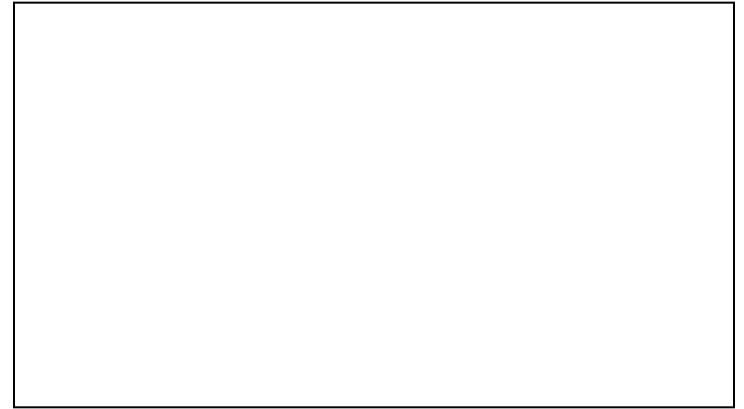
Example of Normal Execution – ให้ค่า step กว้างมากขึ้นก็ได้

for loop to hand-trace

```
for (i = 1;  
     i <= 20;  
     i *= 2)  
    System.out.print(i+" ");
```



What is the output?



The update can be any expression

Example of Normal Execution – Taking Even Bigger Steps

for loop

```
for (i = 1;  
     i <= 20;  
     i *= 2)  
    System.out.print(i+" ");
```

The output

```
1 2 4 8 16
```

The “step” can be multiplicative or any valid expression

End Skipping

Slides will continue.

Confusing Everyone, Most Likely Including Yourself

- A **for** loop is an *idiom* for a loop of a particular form. A value runs from the start to the end, with a constant increment or decrement.
- As long as all the expressions in a **for** loop are valid, the compiler will not complain.

Confusing Everyone, Most Likely Including Yourself

A `for` loop should only be used to cause a loop variable to run, with a consistent increment, from the start to the end of a sequence of values.

Or you could write this (it works, but ...)

```
for (System.out.print("Inputs: ");  
     x = sc.nextInt(); sum+=x)  
{  
    count++;  
}
```

Know Your Bounds – Symmetric vs. Asymmetric

- ค่าเริ่มต้นและค่าสุดท้ายของตัวแปร `loop` ควรจะตอบโจทย์ของเรา
- ช่วง $3 \leq n \leq 17$ เป็นช่วง *symmetric*, เพราะจุดเริ่มและสิ้นสุดถูกรวมไว้:

```
for (n = 3; n <= 17; n++) ...
```

Know Your Bounds – Symmetric vs. Asymmetric

- เมื่อเราจะต้องมาจัดการ **arrays** (ซึ่งจะเรียนภายหลัง), เราจะพบว่า “ถ้า array มีขนาด N เราจะต้องจัดการกับช่วง $[0 \dots N)$ ”
ดังนั้น **for loop** สำหรับการจัดการ arrays คือ:

```
for ( arrIndVar = 0;  
      arrIndVar < N;  
      arrIndVar++ ) ...
```

- Loop จะถูกวน N ครั้ง โดยที่ตัวแปร loop `arrIndVar` มีค่าตั้งแต่ $0, 1, \dots, N-1$

ผู้เขียนโปรแกรมส่วนใหญ่ใช้รูปแบบ *asymmetric* นี้สำหรับทุก ๑ ปัญหา ที่ต้องวน for loop N ครั้ง.

How Many Times Was That?

Fence arithmetic



อย่าลืมนับรวมเลขเสารแรก (หรือสุดท้าย) ที่ `loop` จะต้องนำไปใช้

Fence Arithmetic – Counting Iterations

- การหาขอบล่าง ขอบบน และเงื่อนไขที่ถูกต้อง อาจทำให้สับสนได้
 - เราควรเริ่มจาก 0 หรือจาก 1?
 - เราควรใช้ $\leq b$ หรือ $< b$ เพื่อเป็นเงื่อนไขการจบ loop?
- การนับจำนวน loop จะเป็นสิ่งสำคัญที่จะทำให้เราเข้าใจ loop มากยิ่งขึ้น

Fence Arithmetic – Counting Iterations

การนับ loop ที่เป็นขอบแบบไม่สมมาตรนั้นง่ายกว่า

The loop

```
for (i = a; i < b; i++)...
```

จะวนทำ statements $(b - a)$ ครั้ง
และถ้า a เป็น 0 จะวน b ครั้ง.

เช่น, the loop traversing the characters in a **string**,

```
for (i = 0; i < st.length(); i++)...
```

จะวนเท่ากับความยาวของ string ครั้ง.

ทำให้้งานต่อการจัดการ string, เพราะมีจำนวนอักขระอยู่เท่านั้นพอดี

Fence Arithmetic Again – Counting Iterations

loop ที่มีขอบแบบสมมาตร,

```
for (i = a; i <= b; i++) ...
```

จะถูกลวน $(b - a) + 1$ ครั้ง.


" $+1$ " ที่เพิ่มเข้ามาอาจจะเป็นต้นเหตุของ error ได้.

The do Loop

เงื่อนไขของ **while** loop จะเป็นสิ่งที่ถูกตรวจสอบก่อน **statement** อื่น.

แต่ใน **do** loop (หรือ **do-while** loop) เงื่อนไขจะถูกตรวจสอบก็ต่อเมื่อ
ทำกลุ่มของ **statement** ใน **block** เสร็จไปแล้วรอบหนึ่ง.

```
do  
{  
    statements  
}  
while (condition);
```



The do Loop

นี่ก็หมายความว่า **do loop** ควรจะถูกใช้ก็ต่อเมื่อ **statements** ใน **block** ของ **loop** จะถูกทำก่อนที่จะมาดูแลเรื่องเงื่อนไข

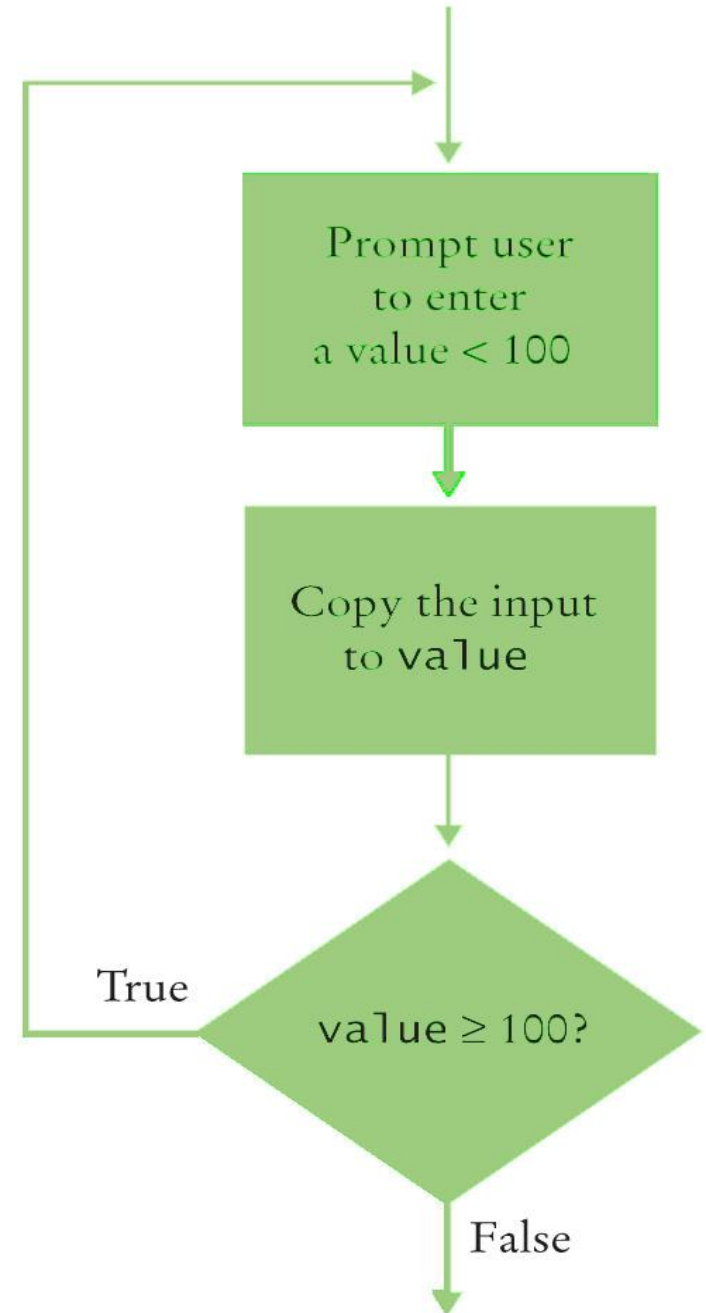
ดังนั้น **statement** ใน **block** ของ **do loop** จะถูกใช้แน่ ๆ อย่างน้อยหนึ่งครั้ง

The do Loop

ปัญหาอะไรที่เราต้องทำอะไรก่อนที่จะตรวจสอบเงื่อนไขของ loop?

เช่น รอรับ **input** จากผู้ใช้จนกว่า **input** จะถูก

ด้านข้างเป็น **flowchart** สำหรับปัญหาที่ผู้ใช้ควรที่จะใส่ค่าน้อยกว่า **100** และ โปรแกรมต้องวนลูปไปเรื่อย ๆ จนกว่า ผู้ใช้จะใส่ค่าให้ถูก



The do Loop

Here is the code:

```
int value;  
do  
{  
    System.out.print("Enter value < 100: ");  
    value = sc.nextInt();  
}  
while (value >= 100);
```

In this form, the user sees the same prompt each time until the enter valid input.

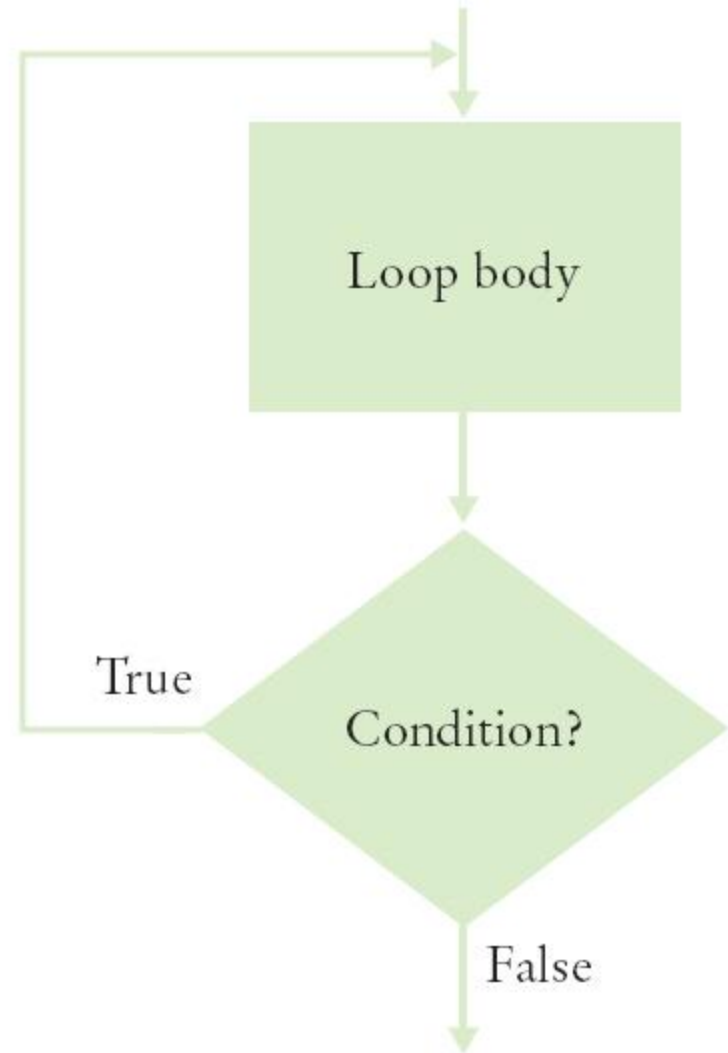
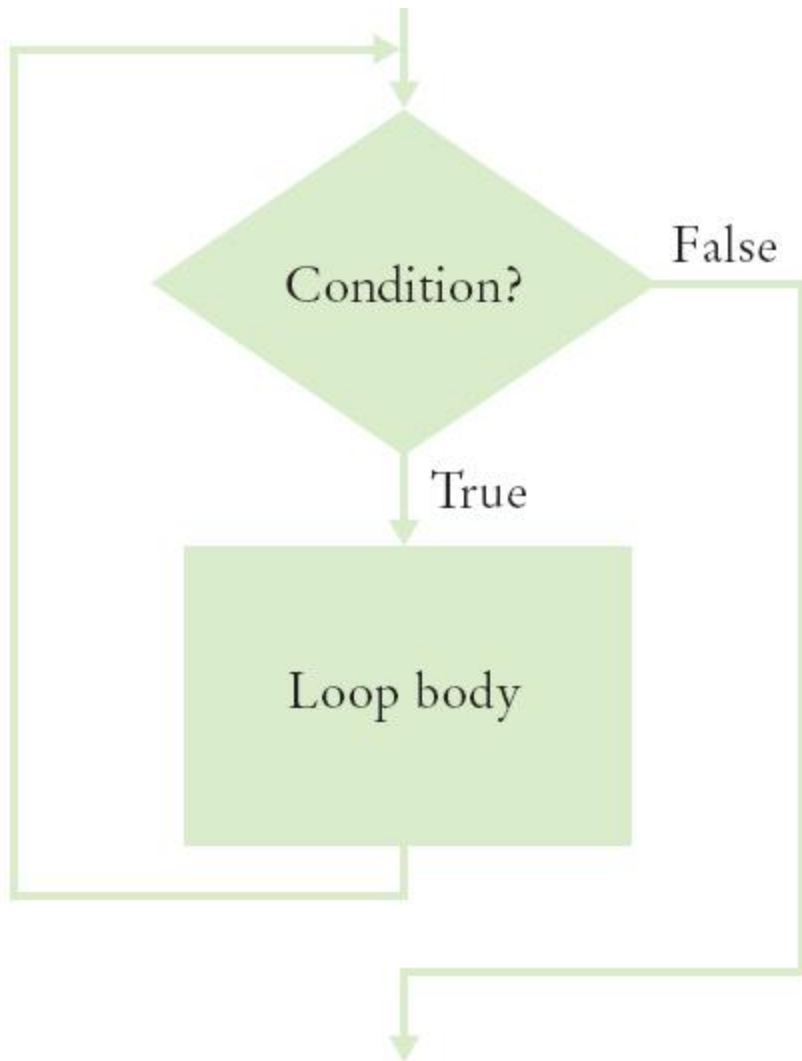
The do Loop

ในกรณีที่เรอยากให้พิมพ์ข้อความที่ต่างกัน เช่น พิมพ์ “error” เฉพาะตอนที่ผู้ใช้ใส่ค่าผิด, ข้อความธรรมดาให้ผู้ใช้ input จะอยู่ก่อน **while** loop:

```
int value;  
System.out.print("Enter a value < 100:");  
while (value >= 100)  
{  
    System.out.println("Sorry, that is larger than 100");  
    System.out.print("Try again: ");  
    value = sc.nextInt();  
}
```

สังเกตถึงการเปลี่ยนแปลงของโปรแกรมจาก slide ที่แล้ว เมื่อ user ป้อนข้อมูลเข้าถูกต้องในขั้นตอนแรก โปรแกรมนี้จะไม่เข้ามาทำงานใน loop -> ดี

Flowcharts for the `while` Loop and the `do` Loop



Processing Input – เราจะหยุด **loop** เมื่อไหร่ และ อย่างไร?

When and/or
how to stop?

Processing Input – When and/or How to Stop?



or be stopped!

Processing Input – When and/or How to Stop?

- ในบางโปรแกรม เราต้องการรู้ว่า ผู้ใช้จะหยุดป้อน input เมื่อไหร่
- เราจะต้องเรียกค่าซึ่งมีความหมายว่า STOP!
- นั่นคือ, เราจะใช้ค่าอะไรก็ได้ที่เรากำหนดว่า ให้มีความหมายว่า ให้หยุด (เรียกว่า ค่า sentinel) ตราบใดก็ตามที่ค่านั้นไม่ใช่ค่าของข้อมูลที่อยู่ในช่วงที่ถูกต้อง

Processing Input – When and/or How to Stop?

- เราจะเขียน **code** ที่คำนวณค่าเฉลี่ยของเงินเดือนที่ถูกผู้ใช้ใส่เข้ามา

ผู้ใช้จะใส่เข้ามากี่ตัว?

นั่นแหละคือปัญหา. เราไม่สามารถเดาได้ว่า ผู้ใช้ใส่เข้ามากี่ตัว.

แต่เราสามารถกำหนด *sentinel value* ได้โดยตกลงกับผู้ใช้ให้ดีกว่า
เมื่อผู้ใช้จะหยุดป้อนข้อมูลให้ใช้ค่านี้

- เพราะเรารู้ว่าเงินเดือนไม่เคยเป็นค่า — เราก็อาจใช้ **-1** เป็นค่า **sentinel** ได้อย่างหายห่วง

Processing Input – When and/or How to Stop?

- เราจะหาค่าเฉลี่ยได้ก็ต่อเมื่อเอาค่าผลรวมของเงินเดือนหารด้วยจำนวนคนทั้งหมด
- แต่ถ้า ผู้ใช้ใส่ค่า **-1** มาแต่แรก ผลรวมของเงินเดือนจะเป็น **0** จำนวนคนที่จะนำมาใช้หารก็เป็น **0** ได้
- เราจำเป็นต้องป้องกัน **error** ชนิดนี้ (dividing by zero) โดยใช้ **if-else** ที่เรียนมา.

The Complete Salary Average Program

ch04/sentinel.cpp

```
public static void main(String [] args)
{
    double sum = 0;
    int count = 0;
    double salary = 0, average;
    // get all the inputs
    System.out.print("Enter salaries, -1 to finish: ");
    while (salary != -1)
    {
        salary = sc.nextDouble();
        if (salary != -1)
        {
            sum = sum + salary;
            count++;
        }
    }
}
```

The Complete Salary Average Program

```
// process and display the average
if (count > 0)
{
    average = sum / count;
    System.out.println("Avg salary: " + average);
}
else
{
    System.out.println("No data");
}

}
```

A program run:

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

the break Statement

- บางครั้งก็ง่ายกว่าถ้าเราบอกผู้ใช้งานว่า “กด Q เพื่อ Quit” แทนที่จะใช้ sentinel value.
- บางครั้งการเลือกค่า sentinel ก็เป็นไปได้ไม่ได้— เพราะข้อมูลอาจเป็นไปได้ทุกค่าเลย เลยไม่สามารถเลือกค่า sentinel ได้?

break Statement

Using the input attempt directly we have:

```
System.out.print("Enter values, Q to quit: ");  
while (true)  
{  
    value = sc.next().charAt(0);  
    if(value == 'Q') break;  
    // statements here  
}
```

Nested Loops (loop ซ้อน loop)



ในแต่ละชั่วโมง, แต่ละนาฬิกาจะถูกประมวลผล – a nested loop.

Nested Loops

- ส่วนใหญ่เราจะใช้ Nested loops สำหรับข้อมูลในตารางในตารางที่เป็นแถวและหลัก.
- เราใช้ loop ในการประมวลผลแต่ละแถว และในแต่ละแถว ในต้องทำอีก loop ซ้อนอยู่ข้างใน loop แรกเพื่อที่จะประมวลผลแต่ละหลัก
- loop ที่อยู่ด้านในเรียกว่า “inner loop,” loop ที่อยู่ด้านนอกเรียกว่า “outer loop.”

Nested Loops

Write a program to produce a table of powers.
The output should be something like this:

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...
10	100	1000	10000

Nested Loops

- ขั้นแรก เราเริ่มจาก loop ด้านในก่อน
- มี 4 หลัก แต่ละหลักจะแสดงผลของการยกกำลัง. ใช้ x เป็นเลขของหลักที่เรากำลังประมวลผล, เราจะได้ pseudo-code:

```
for n from 1 to 4  
{  
    print xn  
}
```

- ก่อนที่เราจะทำต่อ เราควรทดสอบ **code loop** ด้านในก่อน เพราะเราต้องทำแฉวนี้ให้ถูกก่อนที่จะไปทำทุกแฉว

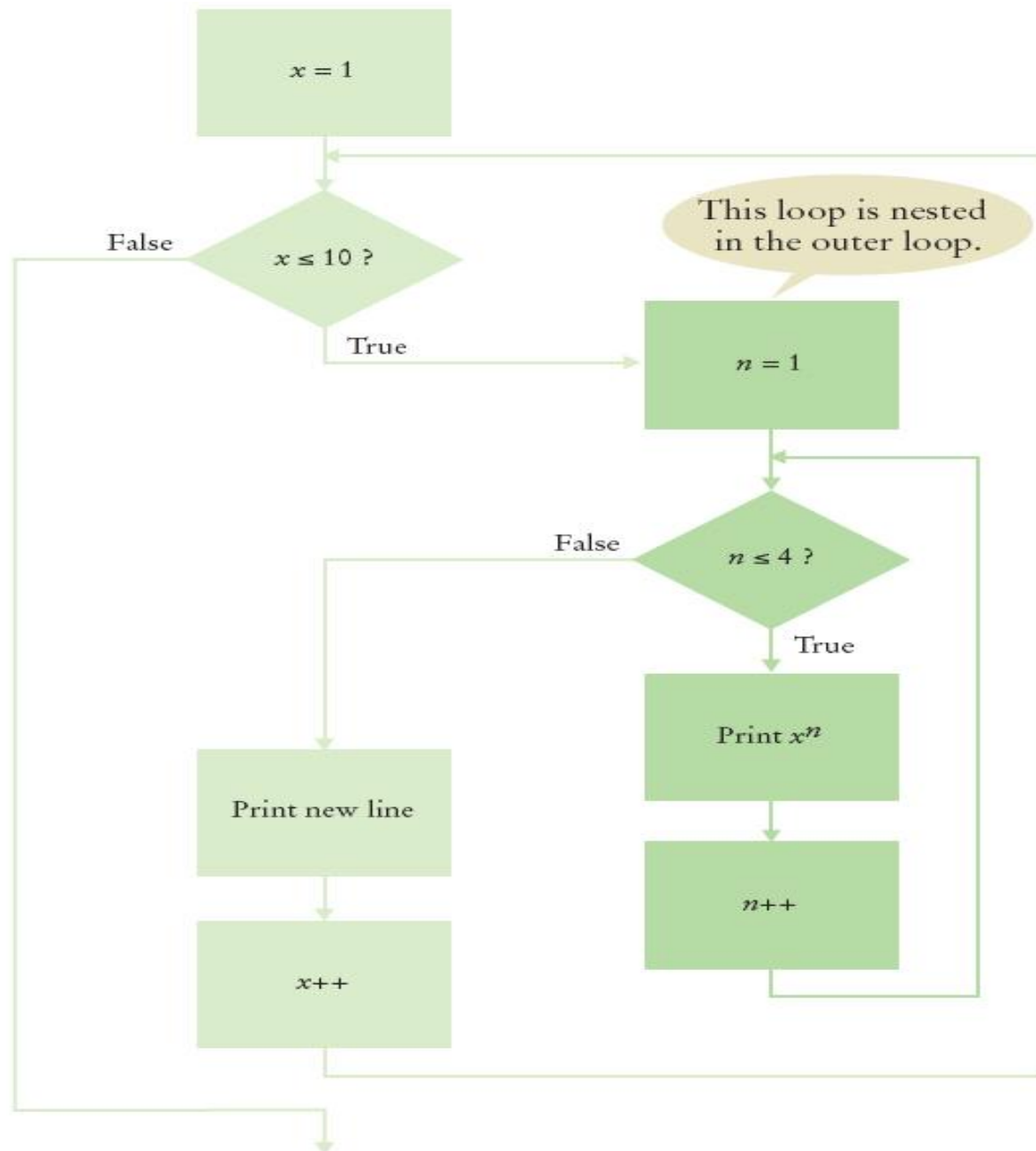
Nested Loops

จากนั้น, ใส่ inner loop เข้าไปใน
outer loop เราจะได้:

(อย่าลืมย่อหน้าด้วย)

```
print table header
for x from 1 to 10
{
    print table row
    print endl
}
```

Nested Loops



The Complete Program for Table of Powers

ch04/powtable.cpp

```
public static void main(String [] args)
{
    final int NMAX = 4;
    final double XMAX = 10;
        int n;
    double x;

    // Print table header
    for (n = 1; n <= NMAX; n++)
    {
        System.out.print(n+" ");
    }
    System.out.println();
    for (n = 1; n <= NMAX; n++)
    {
        System.out.print("x ");
    }
    System.out.println("\n");
```

The Complete Program for Table of Powers

```
// Print table body
for (x = 1; x <= XMAX; x++)
{
    // Print table row
    for (int n = 1; n <= NMAX; n++)
    {
        System.out.print(Math.pow(x, n) );
    }
    System.out.println();
}

}
```

The program run would be:

1	2	3	4
x	x	x	x
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625

More Nested Loop Examples

The loop variables can have a value relationship. In this example the inner loop depends on the value of the outer loop.

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("*");  
    System.out.println();
```

The output will be:

```
*  
**  
***  
****
```

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("*");  
    System.out.println();
```

j จะควบคุมจำนวน loop ซึ่งหมายถึงจำนวน * ในบรรทัด ซึ่ง
จำนวน * ในบรรทัดขึ้นอยู่กับค่าใน *i*

i แทนเลขของแถว หรือ เลข
ของบรรทัด

```
*  
* *  
* * *  
* * * *
```

More Nested Loop Examples

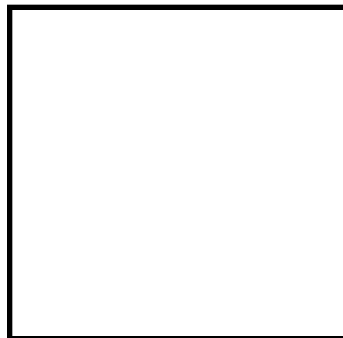
```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("x");  
    System.out.println();
```

j จะควบคุมจำนวน **loop** ซึ่งหมายถึงจำนวน * ในบรรทัด ซึ่ง
จำนวน * ในบรรทัดขึ้นอยู่กับค่าใน **i**

j stops at: i
1

when i is: i 1

i แทนเลขของแถว หรือ เลข
ของบรรทัด



More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("*");  
    System.out.println();
```

j จะควบคุมจำนวน **loop** ซึ่งหมายถึงจำนวน * ในบรรทัด ซึ่ง
จำนวน * ในบรรทัดขึ้นอยู่กับค่าใน **i**

j stops at: i
1

when i is: i 1

i แทนเลขของแถว หรือ เลข
ของบรรทัด

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("*");  
    System.out.println();
```

j จะควบคุมจำนวน **loop** ซึ่งหมายถึงจำนวน ***** ในบรรทัด ซึ่ง
จำนวน ***** ในบรรทัดขึ้นอยู่กับค่าใน **i**

j stops at: i
1

when i is: **i 1**
i 2

*
* *

i แทนเลขของแถว หรือ เลข
ของบรรทัด

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        System.out.print("*");  
    System.out.println();
```

j จะควบคุมจำนวน loop ซึ่งหมายถึงจำนวน * ในบรรทัด ซึ่ง
จำนวน * ในบรรทัดขึ้นอยู่กับค่าใน ***i***

j stops at: *i* *i* *i*
 1 2 3

when *i* is: *i* 1
 i 2
 i 3

*		
*	*	
*	*	*

i แทนเลขของแถว หรือ เลข
ของบรรทัด

More Nested Loop Examples

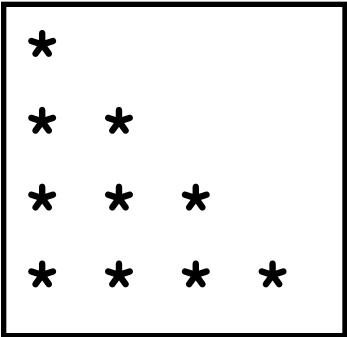
```
for (i = 1; i <= 4; i++)
    for (j = 1; j <= i; j++)
        System.out.print("*");
    System.out.println();
```

j จะควบคุมจำนวน loop ซึ่งหมายถึงจำนวน * ในบรรทัด ซึ่งจำนวน * ในบรรทัดขึ้นอยู่กับค่าใน i

j stops at: i i i i
 1 2 3 4

when i is: i 1
 i 2
 i 3
 i 4

i แทนเลขของแถว หรือ เลขของบรรทัด



More Nested Loop Examples

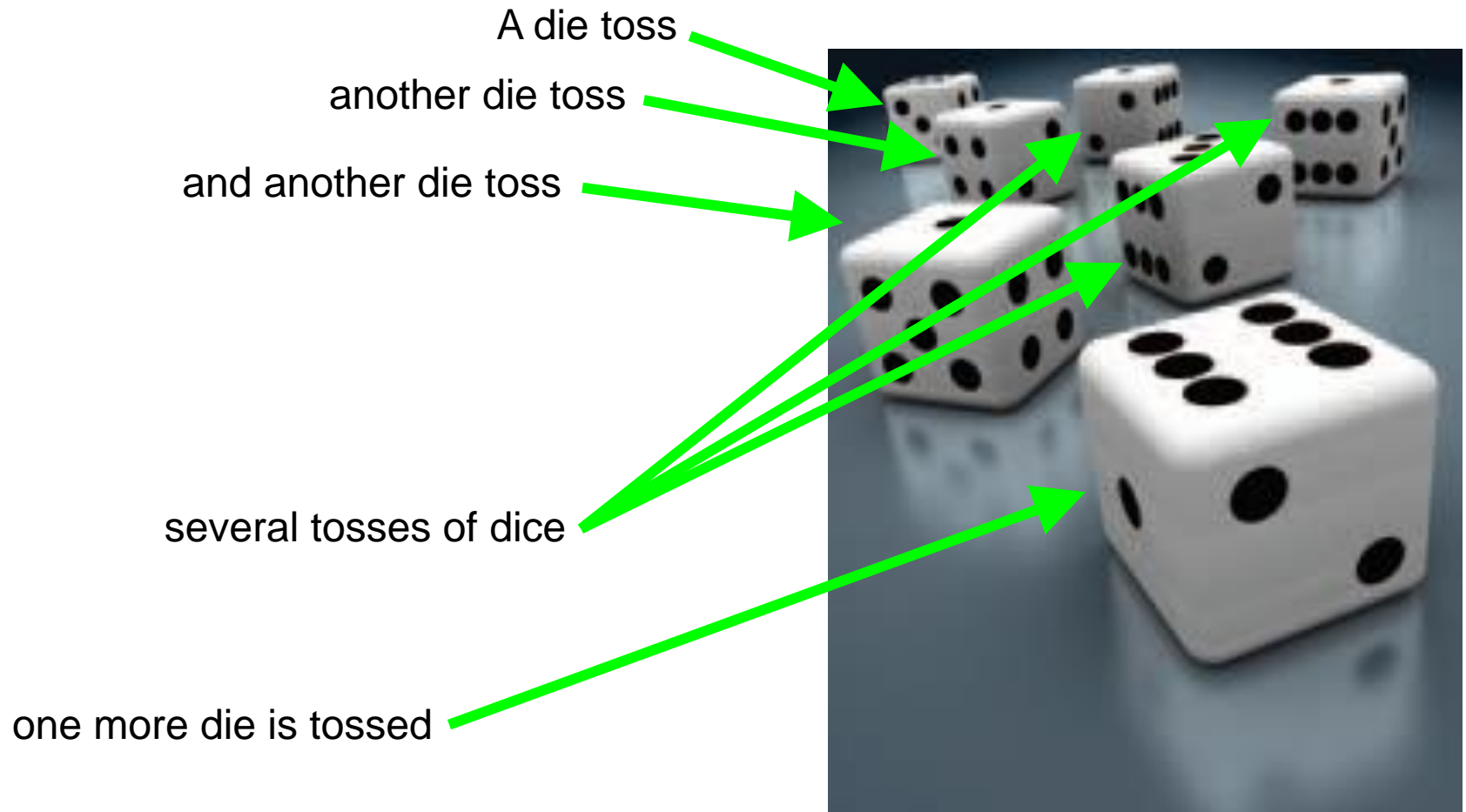
In this example, the loop variables are still related, but the processing is a bit more complicated.

```
for (i = 1; i <= 3; i++)
{
    for (j = 1; j <= 5; j++)
    {
        if (i + j % 2 == 0) {
            System.out.print("*");
        } else {
            System.out.print(" ");
        }
    }
    System.out.println();
}
```

The output will be:

```
* * *
  * *
* * *
```

Random Numbers and Simulations



was that an English lesson?

Simulations

A simulation program uses the computer to simulate an activity in the real world (or in an imaginary one).

Simulations

- Simulations are commonly used for
 - Predicting climate change
 - Analyzing traffic
 - Picking stocks
 - Many other applications in science and business

Randomness for Reality (Simulating)

- Programmers must model the “real world” at times.
- Consider the problem of modeling customers arriving at a store.

Do we know the rate?

Does anyone?

How about the shopkeeper!

Randomness for Reality (Simulating)

Ask the shopkeeper:

*It's about every five minutes
...or so...
...give or a take a couple...
...or three...
...but on certain Tuesdays...*



Randomness for Reality (Simulating)

To accurately model customer traffic, you want to take that random fluctuation into account.

How?

The rand Function

The C++ library has a random number generator:

rand()

The `rand` Function

`rand` is defined in the `cstdlib` header

Calling `rand` yields a random integer
between 0 and `RAND_MAX`

(The value of `RAND_MAX` is implementation dependent)

The `rand` Function

Calling `rand` again yields a different random integer

Very, very, very rarely it might be the same random integer again.

(That's OK. In the real world this happens.)

The `rand` Function

`rand` picks from a very long sequence of numbers that don't repeat for a long time.

But they do eventually repeat.

These sorts of “random” numbers are often called *pseudorandom numbers*.

The `rand` Function

`rand` uses only one pseudorandom number sequence and it always starts from the same place.

Oh dear

The `rand` Function

When you run your program again on another day, the call to `rand` will start with:

the same random number!

Is it very “real world” to use the same sequence over and over?

No, but it’s really nice for testing purposes.

but...

Seeding the `rand` Function

You can “seed” the random generator to indicate where it should start in the pseudorandom sequence

Calling `srand` sets where `rand` starts

`srand` is defined in the `cstdlib` header

Seeding the `rand` Function

But what value would be different every *time* you run your program?

(hint)

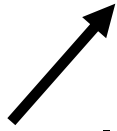


How about the time?

Seeding the `rand` Function

You can obtain the system time.

Calling `time(0)` gets the current time



Note the zero. It is required.

`time` is defined in the `time` header

Seeding the `rand` Function

Calling `srand` sets where `rand` starts.

Calling `time(0)` gets the current time.

So, to set up for “really, really random”
random numbers on each program run:

```
srand(time(0)); // seed rand()
```

(Well, as “really random” as we can hope for.)

Modeling Using the `rand` Function

Let's model a pair of dice,



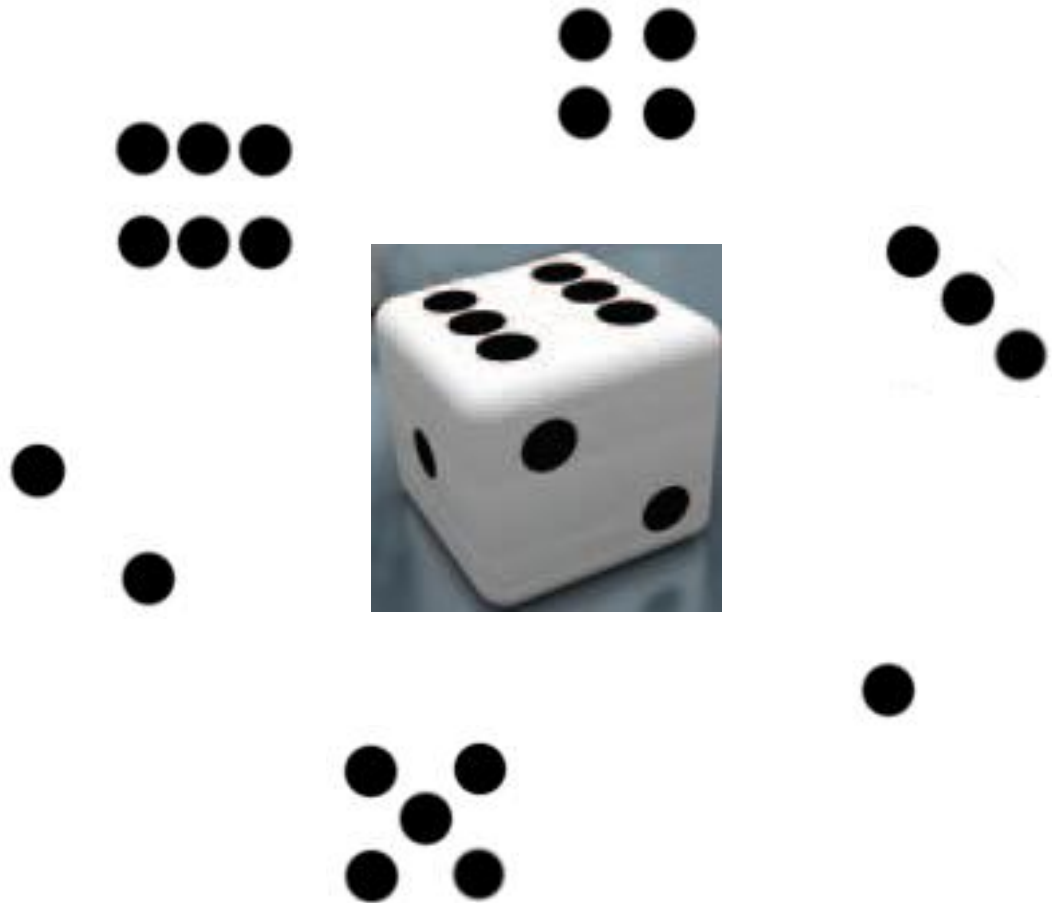
Modeling Using the `rand` Function



one die at a time.

Modeling Using the `rand` Function

What are the numbers on one die?



Modeling Using the `rand` Function

Numbers we can work with please!

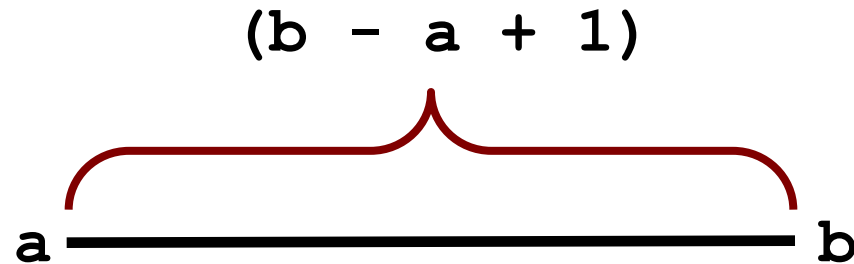
Modeling Using the `rand` Function

What are the bounds of the range of numbers on one die?
1 and 6 (inclusive)



We want a value randomly between those endpoints
(inclusively)

Modeling Using the `rand` Function

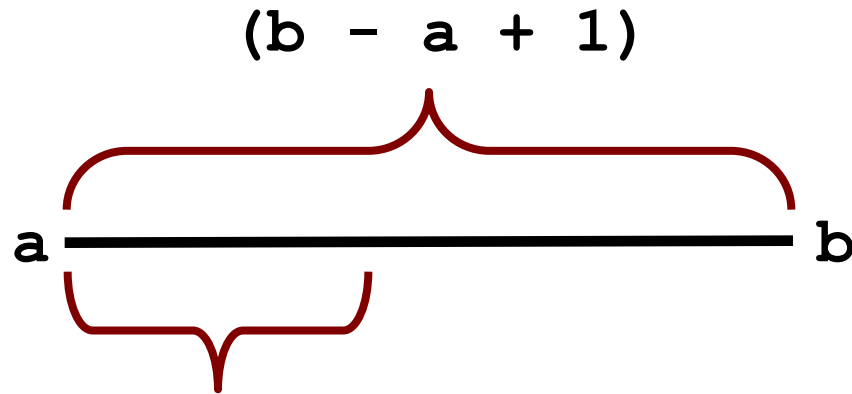


Given two endpoints,
a and **b**, recall there are

$$(b - a + 1)$$

values between **a** and **b**,
(including the bounds themselves).

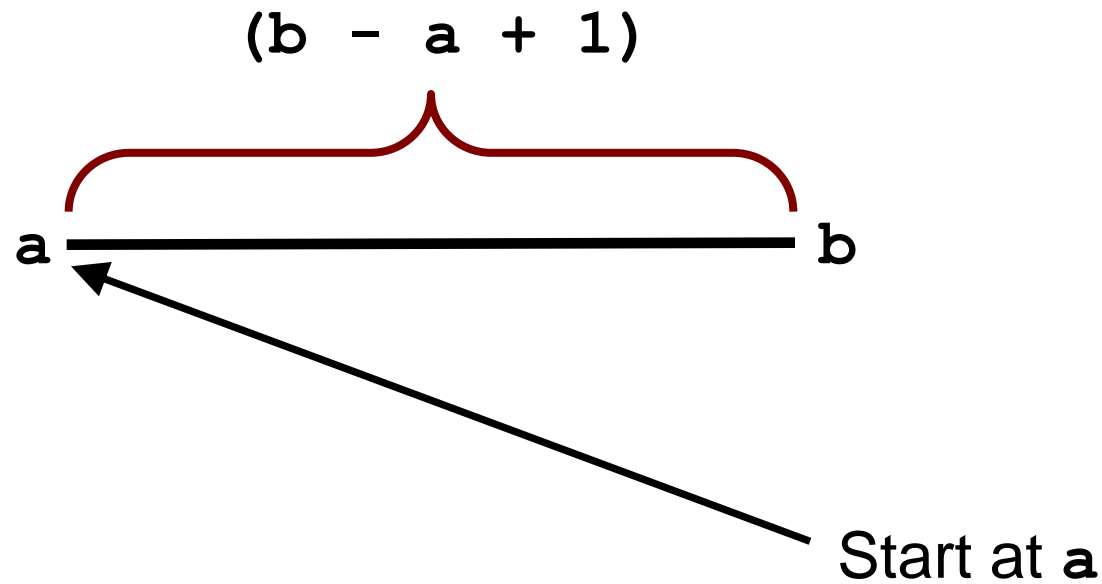
Modeling Using the `rand` Function



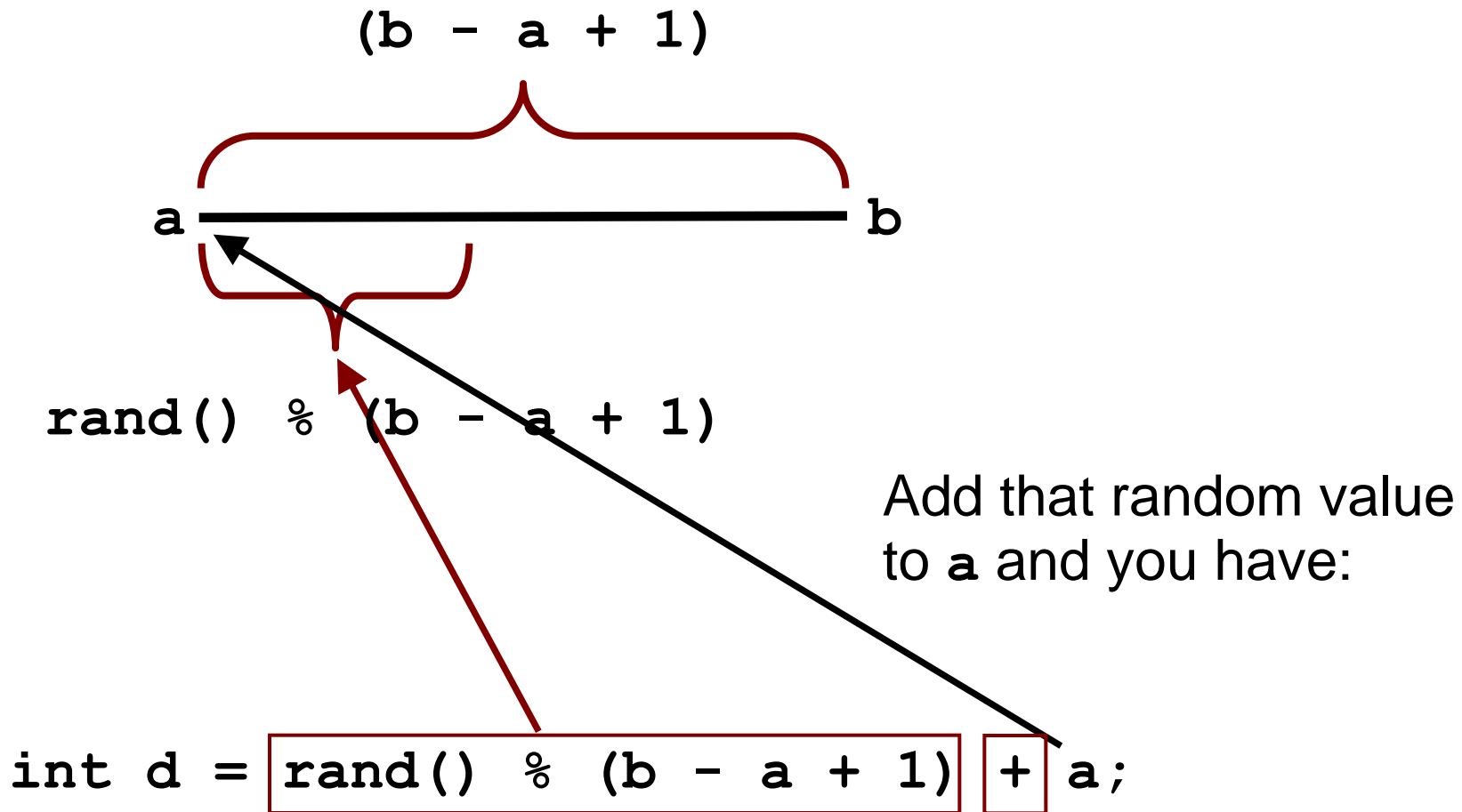
`rand() % (b - a + 1)`

Obtain a random value
between 0 and $b - a$
by using the `rand()` function

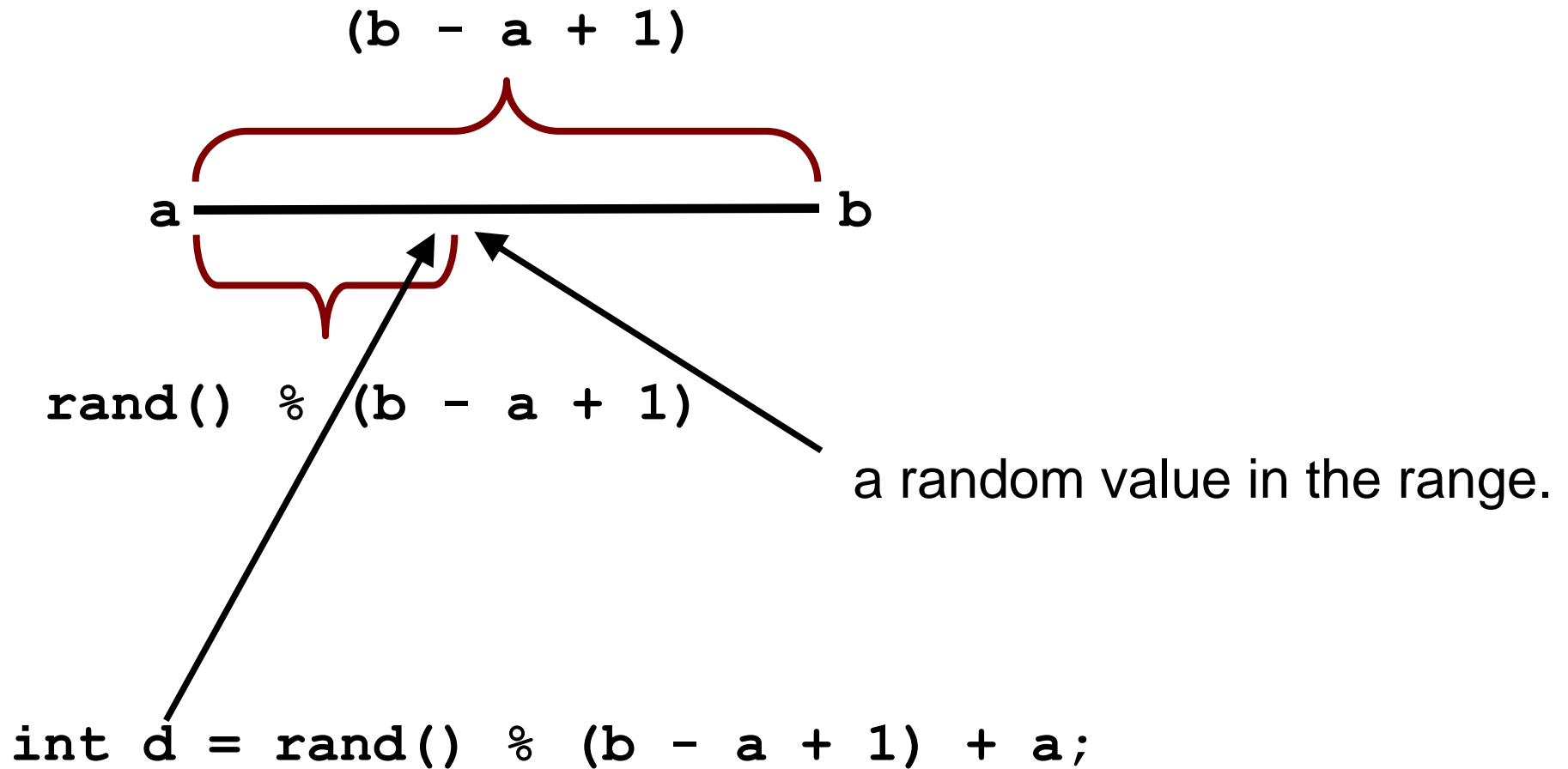
Modeling Using the rand Function



Modeling Using the rand Function



Modeling Using the rand Function



Modeling Using the `rand` Function



Using 1 and 6 as the bounds
and
modeling for two dice,
running for 10 tries,

we have:

Modeling Using the rand Function

ch04/dice.cpp

```
public static void main(String [] args)
{
    srand(time(0)) ;

    for (i = 1; i <= 10; i++)
    {
        int d1 = rand() % 6 + 1;
        int d2 = rand() % 6 + 1;
        System.out.println(d1+" "+d2) ;
    }
    System.out.println() ;

}
```

One of many different
Program Runs:

5	1
2	1
1	2
5	1
1	2
6	4
4	4
6	1
6	3
5	2

The Monte Carlo Method



The premier gaming “*table d’darts*”
at one of the less well known casinos in Monte Carlo,
somewhat close but not quite next door to Le Grand Casino.

The Monte Carlo Method



As long as we're here, let's go in!

The Monte Carlo Method

The Monte Carlo method is a method for finding approximate solutions to problems that cannot be precisely solved.

Here is an example: compute π

This is difficult.

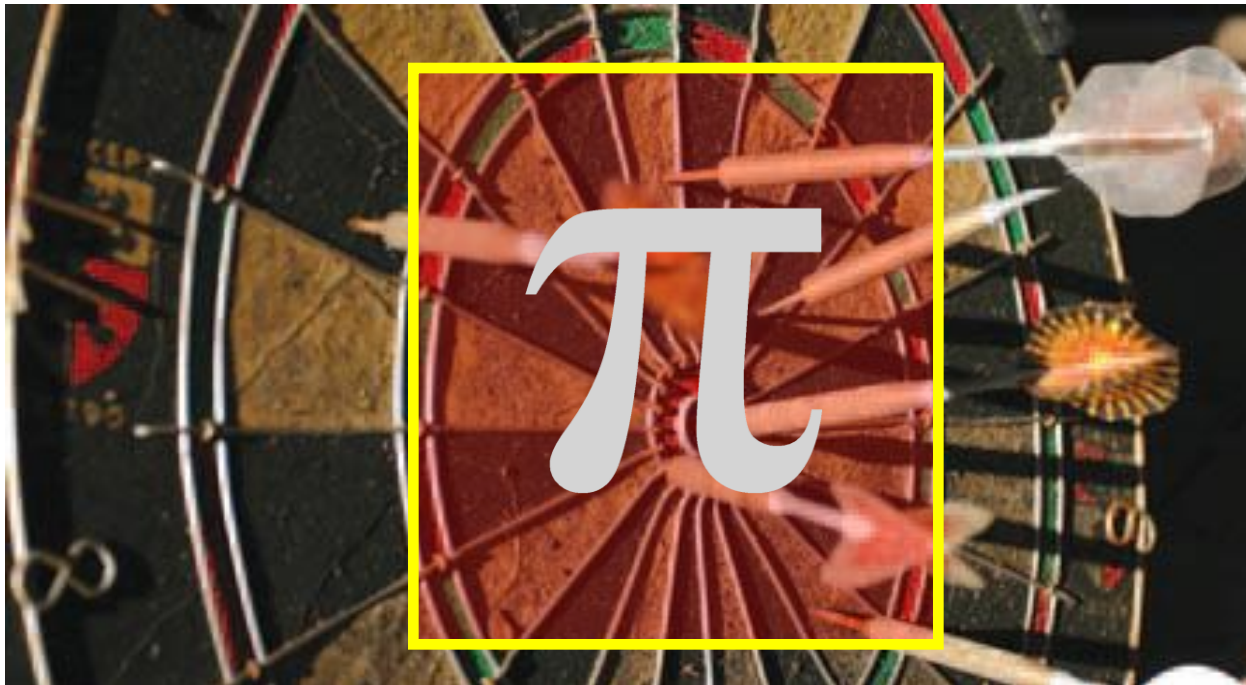
The Monte Carlo Method

While we are in this fine casino,
we should at least play one game at the “*table d’darts*”



The Monte Carlo Method

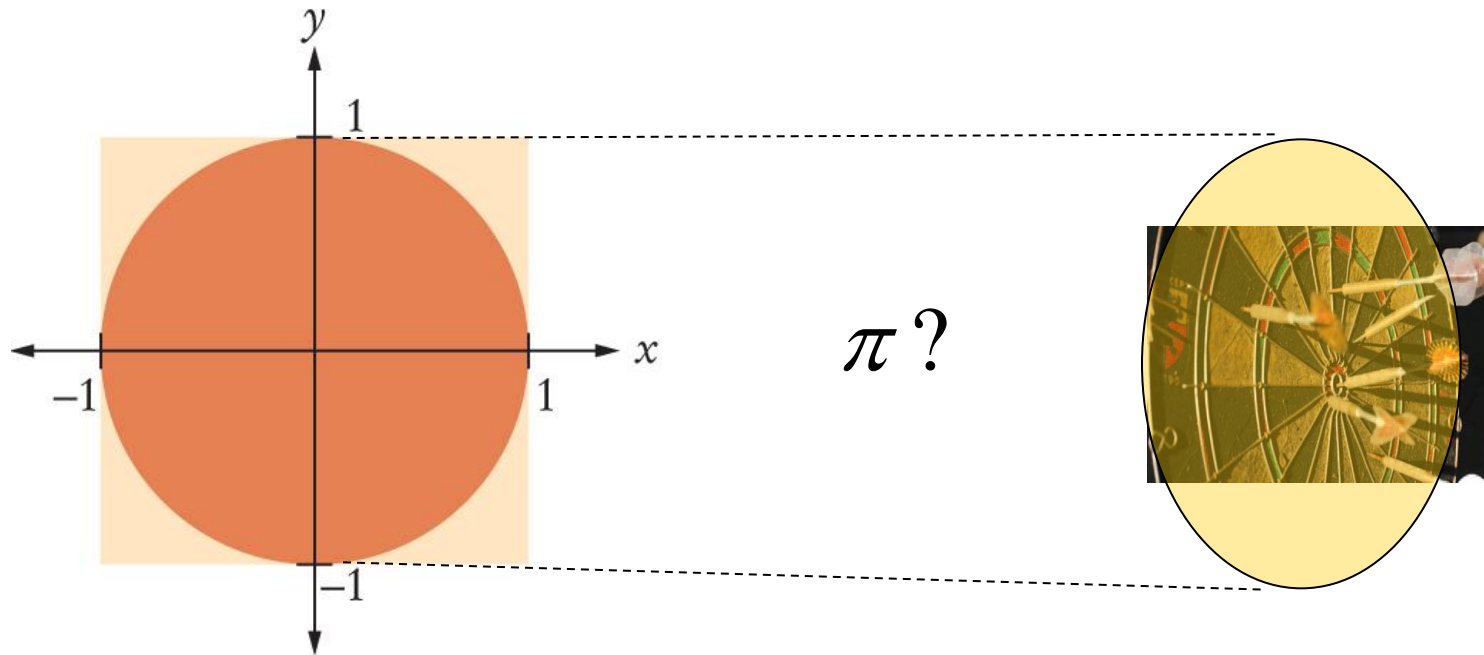
THAT'S IT!



By shooting darts (and a little math)
we can obtain an approximation for π .

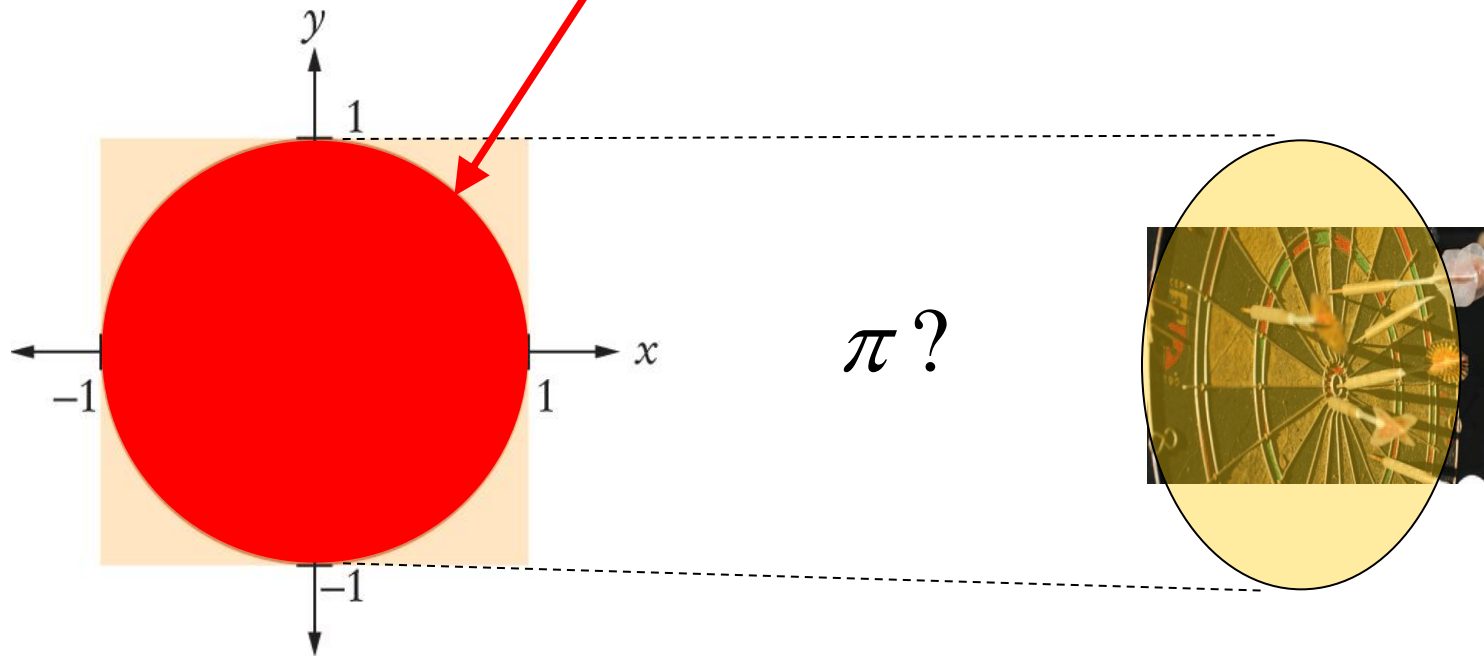
The Monte Carlo Method

Consider placing the round dartboard inside an exactly fitting square



The Monte Carlo Method

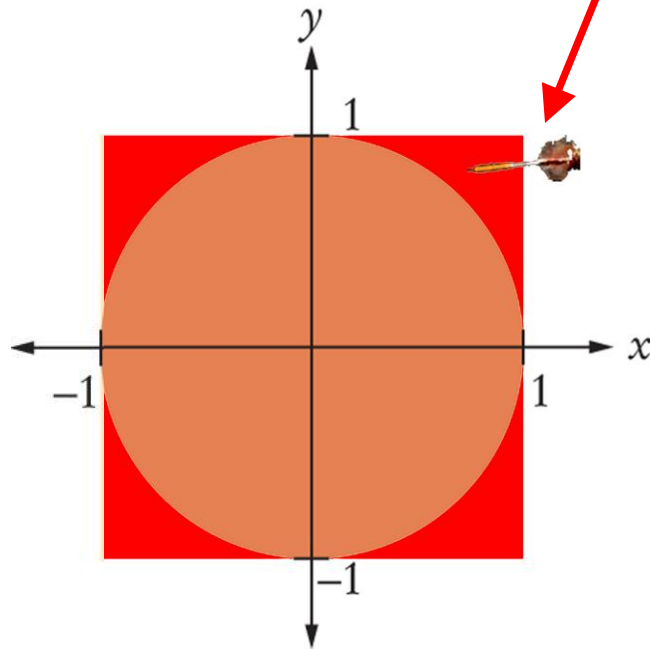
As we toss darts at the target,
if we are able to just *hit* the target – at all – it's a hit.



(no wonder this is such a pathetic casino)

The Monte Carlo Method

and a *miss* is a miss.

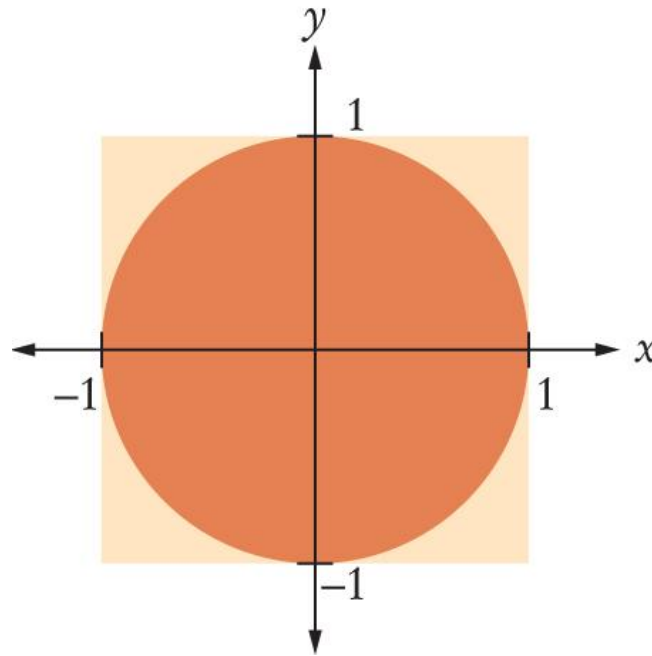


π ?

The Monte Carlo Method

The (x, y) coordinate of a *hit* is when $(x^2 + y^2) \leq 1$.
In code:

```
if (x * x + y * y <= 1) { hits++; }
```



The Monte Carlo Method

Our coded random shots will give a ratio of
hits/tries

that is approximately equal to the ratio of
the areas of the circle and the square:

$$\pi/4$$

The Monte Carlo Method

Multiply by 4 and we have an estimate for π !

$$\pi = 4 * \text{hits/tries};$$

The longer we run our program,
the more random numbers we generate,
the better the estimate.

The Monte Carlo Method

For the x and y coordinates within the circle, we need random x and y values between -1 and 1 .

That's a range of $(-1 + 1 + 1)$ or 2 .

As before, we want to add some random portion of this range to the low endpoint, -1 .

But we will want a floating point value, not an integer.

The Monte Carlo Method

We must use `rand` with `double` values to obtain that random portion.

```
double r = rand() * 1.0 / RAND_MAX;
```

The value `r` is a random floating-point value between 0 and 1.

You can think of this as a percentage if you like.

(Use `1.0` to make the `/` operator not do integer division)

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

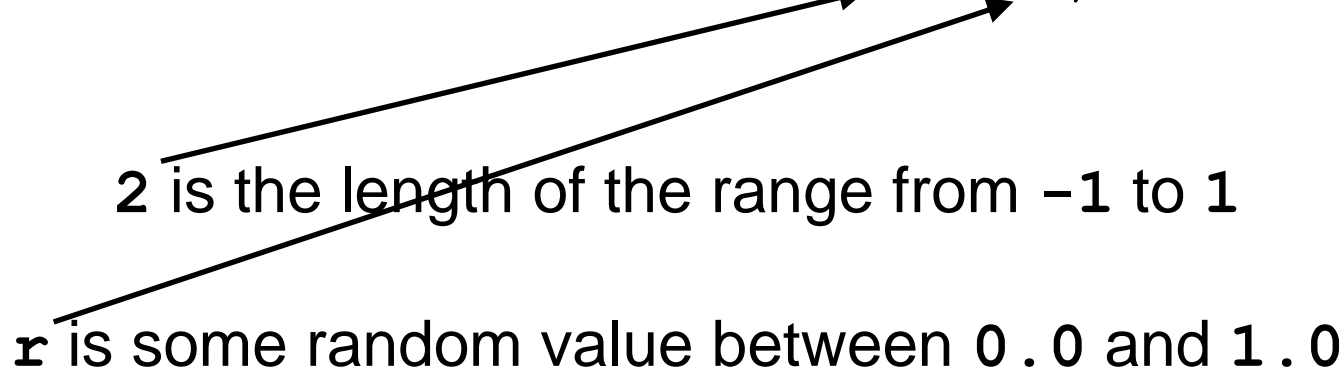
2 is the length of the range from **-1** to **1**



The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```



2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

2 is the length of the range from -1 to 1

`r` is some random value between 0.0 and 1.0

so $(2 * r)$ is some portion of that range

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

so $(2 * r)$ is some portion of that range

We will add this portion to the left hand end of the range, -1

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

so $(2 * r)$ is some portion of that range

Adding this portion to the left hand end of the range gives us:

x randomly within the range -1 and 1.

The Monte Carlo Method for Approximating π

ch04/montecarlo.cpp

```
public static void main(String [] args)
{
    final int TRIES = 10000;
    srand(time(0));
    int hits = 0;
    for (int i = 1; i <= TRIES; i++)
    {
        double r = rand() * 1.0 / RAND_MAX; // Between 0 and 1
        double x = -1 + 2 * r; // Between -1 and 1
        r = rand() * 1.0 / RAND_MAX;
        double y = -1 + 2 * r;
        if (x * x + y * y <= 1) { hits++; }
    }
    double pi_estimate = 4.0 * hits / TRIES;
    System.out.println("Estimate for pi: "+pi_estimate);
}
```

Chapter Summary

Explain the flow of execution in a loop.

- Loops execute a block of code repeatedly while a condition remains true.



- An off-by-one error is a common error when programming loops. Think through simple test cases to avoid this type of error.

Use the technique of hand-tracing to analyze the behavior of a program.

- Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.
- Hand-tracing can help you understand how an unfamiliar algorithm works.
- Hand-tracing can show errors in code or pseudocode.

Use for loops for implementing counting loops.



- The for loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.

Chapter Summary

Choose between the **while loop** and the **do loop**.

- The do loop is appropriate when the loop body must be executed at least once.

Implement loops that read sequences of input data.

- A sentinel value denotes the end of a data set, but it is not part of the data.
- You can use a Boolean variable to control a loop. Set the variable to true before entering the loop, then set it to false to leave the loop.
- Use input redirection to read input from a file. Use output redirection to capture program output in a file.



Use the technique of **storyboarding** for planning user interactions.

- A storyboard consists of annotated sketches for each step in an action sequence.
- Developing a storyboard helps you understand the inputs and outputs that are required for a program.

Chapter Summary

Know the most common loop algorithms.

- To compute an average, keep a total and a count of all values.
- To count values that fulfill a condition, check all values and increment a counter for each match.
- If your goal is to find a match, exit the loop when the match is found.
- To find the largest value, update the largest value seen so far whenever you see a larger one.
- To compare adjacent inputs, store the preceding input in a variable.

Use nested loops to implement multiple levels of iteration.



- When the body of a loop contains another loop, the loops are nested. A typical use of nested loops is printing a table with rows and columns.

Apply loops to the implementation of simulations.

- In a simulation, you use the computer to simulate an activity. You can introduce randomness by calling the random number generator.





End Chapter Four