“Nuts and Bolts” (*)

covering

** variables
** assignment
** keywords
** primitive types
** basic operations
** control structures

Chapters 3–5 (sections 3.1–3.3; 4.1–4.4; 5.1–5.4) – “Big Java” book
Chapters 1+3 – “Head First Java” book
Chapters 2+3 – “Introduction to Java Programming” book
Chapter 2 (sections 2.1 – 2.5) – “Java in a Nutshell” book

(*) Basic, practical details of Java.

Java Program Structure

Basic Program Template:

```
class ClassName {
   public static void main(String[] args) {
      // declarations of variables and methods
      // intermingled with statements
   }
}
```

Example: a simple operation (1/2)

```
/**
 *  A simple operation.
 */
public class MyOperation {
   public static void main(String[] args) {
     int a = 6;
     int b = 5;
     int product = a * b;
     System.out.println("a=" + a);
     System.out.println("b=" + b);
     System.out.println("product=" + a + "*" + b + "=" + product);
   }
}
```

```
a=6
b=5
product = 6*5=30
```

Example: a simple operation (2/2)

```
/**
 *  A simple operation.
 */
public class MyOperation {
   public static void main(String[] args) {
     int a = 6;
     int b = 5;
     int product = a * b;
     System.out.println("a=" + a);
     System.out.println("b=" + b);
     System.out.println("product=" + a + "*" + b + "=" + product);
   }
}
```

data type
variables
constants
expressions

```
a=6
b=5
product = 6*5=30
```

Variables

- Unlike almost every other language, variables in Java can be declared anywhere in a program (though the program logic may require you to define certain variables before certain statements).

- The format for variable declarations is essentially the same as in C.

- Thus you can either declare variables as follows:

Variable Declaration
`typeName name1, name2, ... namen;` `typeName name1 = initvalue;`

Basic types

- Java is strongly typed and strongly classed.
  - Only variables with the same types or classes can be used together.

Primitive Types	
`byte, short, int, long:`	for integers
`float, double:`	for real numbers
`boolean:`	for boolean values
`char:`	for characters

- Every type in Java has a default value, and sometimes a variable will be initialised to this default value.

Always get in the habit of initialising your variables.
We will see later when Java uses default values and when it does not.

What about Strings?
In Java, Strings are not a primitive data type; they are objects. More about this later …

integers

Java provides the following basic integer types:

- **`byte`**

  A small integer, in the range -128 to +127.
  Default value: **`0`**

- **`short`**

  A small integer, in the range -32768 to +32767.
  Default value: **`0`**

- **`int`**

  Integers. A value in the range -2147483648 to +2147483647
  Default value: **`0`**

- **`long`**

  A large integer. In the range -10^{18} to $+10^{18}$
  Default value: **`0L`**

More primitive data types ...

- **`float`**

  A floating point value (so, not very precise).

  Default value **`0.0f`**

- **`double`**

  A double precision floating point value (i.e. more precise)
  Default value is **`0.0d`**

- **`boolean`**

  Either **`true`** or **`false`**.
  Default value: **`false`**

- **`char`**

  A character; not ASCII character, but UNICODE (a 16-bit international standard encoding for portability).
  Default value: **`u0000`**

Primitive Data Types: Quick Reference

Type	Representation	Initial value	Storage	Max. value
`byte`	signed integer	0	8 bits	127
`short`	signed integer	0	16 bits	32767
`int`	signed integer	0	32 bits	2147483647
`long`	signed integer	0	64 bits	over 10^{18}
`float`	floating point	0	32 bits	over 10^{38}
`double`	floating point	0	64 bits	over 10^{308}
`boolean`	true or false	false	1 bit	
`char`	UNICODE (not ASCII)	u0000	16 bit	uFFFF

Be Careful Bears Shouldn't Ingest Large Furry Dogs! ☺

Naming Guidelines

- The rules for variable names (and indeed all identifiers) follow the same conventions as in C:
  - The convention is to intercap names, e.g.
    `staffSalary` is preferred to `staff-salary`
  - Variables and methods should start with lowercase:

```
int x;
int myVariable = 0;
public void myMethod();
```

  - Class names should start with uppercase:

```
public class Example {}
public class MyFirstJavaProgram {}
```

  - Use pronounceable names.
  - Use names that are descriptive (but be brief), e.g.
    `calculatePower()` or `calcPower()`,
    not `calculateThePowerOfTheInputVariable()`

Variable Names: Good Examples

```
int anInteger = 42;
byte smallNumber = 2;
short shortNumber = 1234;
long aVeryLongNumber = 86827263927L;
float ratio = 0.2363F;

double delta = 453.523311903;
char topGrade = 'A';
char another = 'c';
boolean flag = true;
boolean done = false;
```

If you write `long aVeryLongNumber = 86827263927`, Java "interprets" this as an `int` variable.

If you write `float ratio = 0.2363`, Java "interprets" this as a `double` variable.

… and things for you to try out!

Java Reserved Words

- Every programming language has keywords that cannot be used as identifiers; so does Java. Some of the Java keywords are:

abstract	else	interface	super
boolean	extends	**long**	switch
break	final	native	synchronized
byte	finally	new	this
case	**float**	null	throw
catch	for	package	throws
char	goto	private	transient
class	if	protected	try
const	implements	**public**	**void**
continue	import	return	volatile
do	instanceof	**short**	while
double	**int**	**static**	

Assignments and Operators (1/2)

- As in C, assignments are done via the = statement

```
int i;
i = 9;
i = i + 1;        // i has the value 10
```

- Java also provides the ++ and -- operators to increment and decrement an int variable by 1:

```
int i;
i = 9;     // i has the value 9
i++;       // i has the value 10
i--;       // i has the value 9
```

Assignments and Operators (2/2)

```
i++;
```

is essentially the same as

```
i = i + 1;
```

Increments and Decrements	
postincrement: `i++`	preincrement: `++i`
postdecrement: `i--`	predecrement: `--i`

- A post operation causes the variable to be used 'as is' in the current statement, and it is incremented or decremented afterwards.
- A pre operation causes the variable to first be incremented or decremented, and then it is used in the current statement.

Try to use only post operations to begin with!

Operators

- Operators for numeric values in Java are essentially the same as in C:

Operators	
+	addition
*	multiplication
%	remainder
-	subtraction
/	division

- Examples:

```
int i, j, k;
i = 9;
j = 3;
k = 2;
i = i * (j + k);      // i=?
j = i / 4;            // j=?
k = i % 4;            // k=?
```

Arithmetic Assignment Operators

- All arithmetic operators can be combined with assignment statements.

- Examples:

```
int c=3;
c += 7;          // c = c + 7 = ?
c -= 5;          // c = c - 5 = ?
c *= 6;          // c = c * 6 = ?
c /= 3;          // c = c / 3 = ?
c %= 3;          // c = c % 3 = ?
```

… and things for you to try out!

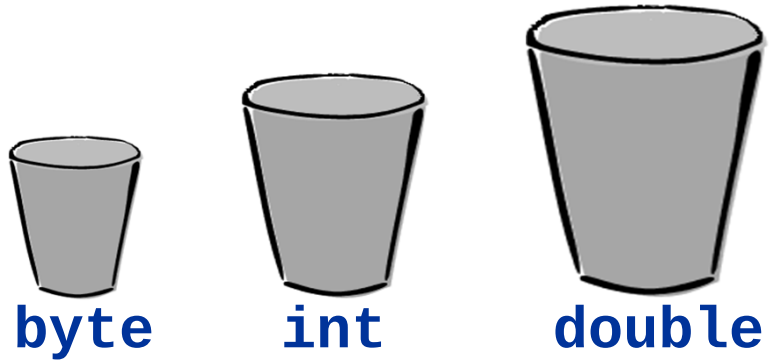
Type Cast and Casting Primitive Variables

- Conversion between numeric types:

  `byte => short => int => long => float => double`

- In the other direction, e.g. `float => int`, we have to use the type cast operator.

Type Cast
(type) expression



- When a variable is declared, it is like a cup has been created for that variable of exactly the right size and shape.
- It can only ever be of that size and shape:
  - it cannot change
  - the only things that can use that cup must be of that size and shape!

Example: Type Cast

```
/**
  * ExampleTypeCast: prints an example of type casting
  * @author Raul Mondragon
  */
class ExampleTypeCast {
  public static void main(String[] args) {
     double x = 6.8;
     int i, j;
     j = (int)(x + 1.3);
     i= (int) x + (int) 1.3;
     System.out.println("j = " + j + "," + "i = " + i);
  }
}
```

The output is:
`j = 8,i = 7`

Operator Precedence

Equal Precedence

High Precedence

Low Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
conditional	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

More information at:
http://download.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Other Precedence Rules

- Operators with higher precedence are evaluated before operators with relatively lower precedence.

- Operators on the same line have equal precedence.

- Equal precedence:
  - binary operators (except assignment) are evaluated from left to right
  - assignment operators are evaluated right to left

Precedence Examples 1+2

```
int a = 4;
int b = 1;
int c = 3;
int result = a - b + c;
System.out.println(result);
```

both operators are of equal precedence in our table

∴ evaluate left to right!
`result = 6` (not **`0`**)

```
int a = 4;
int b = 1;
int c = 3;
int result = a = b = c;
System.out.println(result);
```

both operators are of equal precedence in our table

∴ evaluate <u>right to left</u>!
`result = 3`

Precedence Examples 3 – 6

Post incrementing

```
int a = 4;
int result = a++ + a;
System.out.println("result = " + result + ", a = " + a );
```

result = ? a = ?

```
a = 4;
result = a + a++;
System.out.println("result = " + result + ", a = " + a );
```

result = ? a = ?

Pre incrementing

```
int a = 4;
int result = ++a + a;
System.out.println("result = " + result + ", a = " + a );
```

result = ? a = ?

```
a = 4;
result = a + ++a;
System.out.println("result = " + result + ", a = " + a );
```

result = ? a = ?

Precedence Operators: Advice

- The examples from the two previous slides may behave differently in other languages!
- For example, never say something like:

```
int i = 0;
i = i++;     // now i is 0, not 1
```

- Keep it simple: try to use `++` and `--` only in standalone statements, not in expressions.

- You will be expected to be able to decipher (examples of this):

`int x = a++ + 5;` but not `int x = a++ + ++a;`

- Similarly, avoid mixing assignment and truth tests or other statements:

```
if ((a=5) == b)
  int k = j + (a=5);
```

Brackets

- Always use brackets to make your conditions more readable.
- Example:

  `a + b * c` is more readable as

  `a + (b * c)`

- You should use brackets even if you want to follow operator precedence rules.

Exercises

5 minutes

1. Which of the following are NOT Java keywords?

   `class, public, thrown, x, extends`

2. Write Java statements to do the following:
   - Declare an `int` variable `count` with initial value `10`.
   - Add `10` to it, and create a new variable `result` that is equal to `count*count`.
   - Set variable `count` equal to `count` divided by `4`.

   What are the variables `count` and `result` equal to?

Control Structures and Relational Operations

- Java provides all the obvious control structures:
  - Selection
    - `if ...`
    - `if ... else ...`
    - `switch`
  - Repetition
    - `while ...`
    - `do ... While ...`
    - `for`

Relational Operators	
>	greater than
<	less than
<=	less than or equal to
>=	greater than or equal to
==	equality test
!=	unequality test

Don't use = where you mean ==.

Logical operators: and, or, not

- Java also provides “and”, “or”, and “not” operators:

&&	and
||	or
!	not

- Examples:

&&

true && true	true
true && false	false
false && true	false
false && false	false

||

true || true	true
true || false	true
false || true	true
false || false	false

!

!true	false
!false	true

`if` and `if-else` statements

```
public class Grade {
   public static void main(String[] args) {
      int grade = 70;
      if (grade >= 40) {
         System.out.println("passed");
      }
   }
}
```

Relational expression: it must evaluate to a **`boolean`** value.

```
public class Grade {
   public static void main(String[] args) {
      int grade = 30;
      if (grade >= 40) {
         System.out.println("passed");
      }
      else {
         System.out.println("failed");
      }
   }
}
```

Common error & things to be careful of

```
int a = 1;
int b = 0;

if (a = b) {
  // action ...
}
else {
  // action ...
}
```

`if(a==b)`

- Example:

```
if ((j>i) && (j<k) && (j<=j) && (i++>4)) {
  System.out.println("no");
}
```

Use brackets (parentheses) to make your conditions readable!

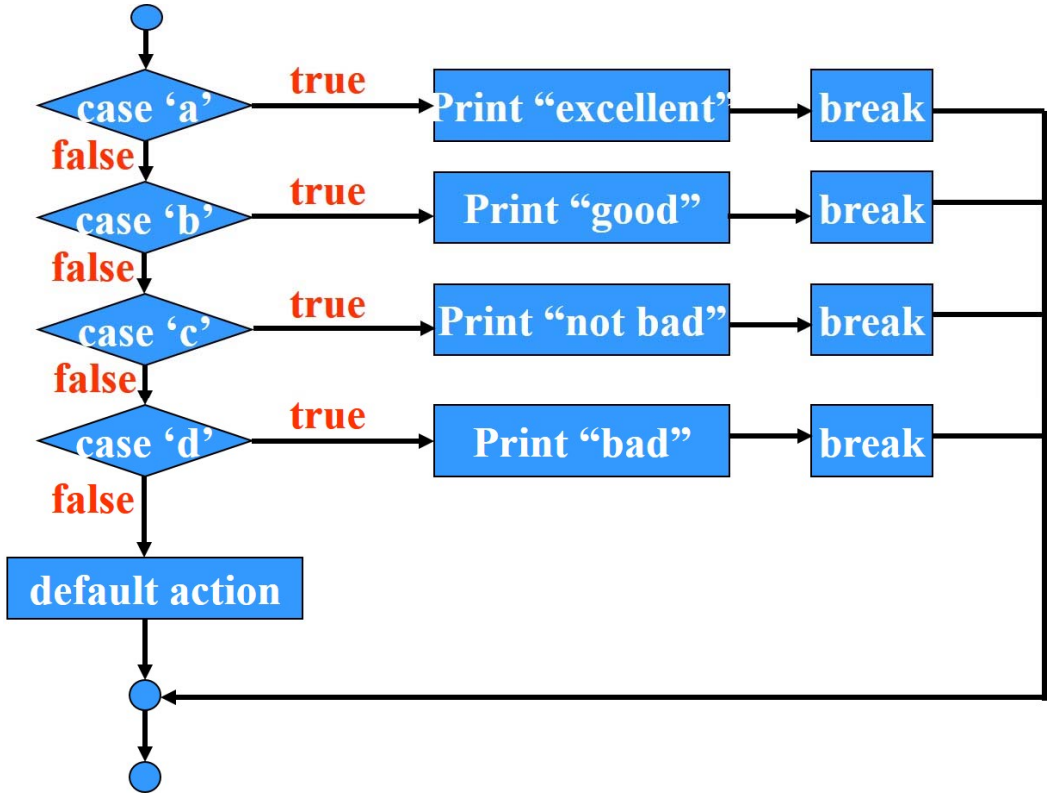
Nested `if-else` statements

```
if (grade >= 70) {
  System.out.println("Grade A");
} else if (grade >= 60) {
    System.out.println("Grade B");
  } else if (grade >= 50) {
      System.out.println("Grade C");
    } else if (grade >= 40) {
        System.out.println("Grade D");
      } else {
          System.out.println("Grade E");
        }
```

… and things for you to try out!

The switch statement

```
char grade = 'a';
switch (grade) {
  case 'a':
    System.out.println("excellent");
    break;
  case 'b':
    System.out.println("good");
    break;
  case 'c':
    System.out.println("not bad");
    break;
  case 'd':
    System.out.println("bad");
    break;
  default:
    System.out.println("no such grade!");
}
```



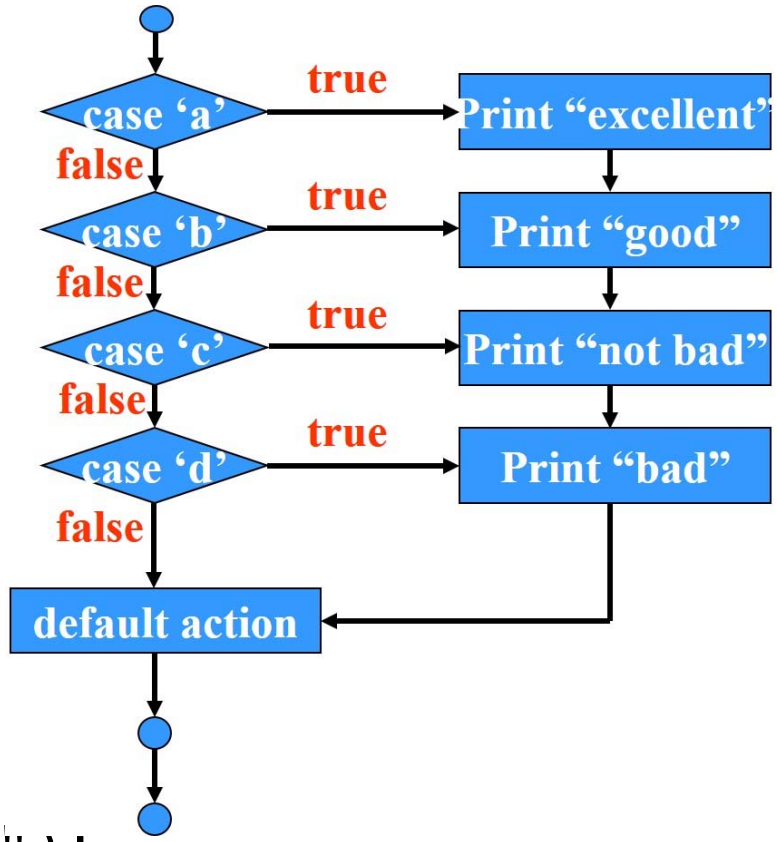
break – causes the remainder of the **switch** statement to be skipped

default – action in case none of the cases match

Example: switch with missing breaks

```
char grade = 'a';

switch (grade) {
  case 'a':
    System.out.println("excellent");
  case 'b':
    System.out.println("good");
  case 'c':
    System.out.println("not bad");
  case 'd':
    System.out.println("bad");
  default:
    System.out.println("no such grade!");
}
```




```
excellent
good
not bad
bad
no such grade
```

Conditional Operator

Conditional Operator
? : shorthand **if-then**

```
a = (test? b:c);  // a=b if test is true,
                  // else a=c.
```

Another example:

```
int total =10;
total = (total > 5) ? total+1 : total*2;
```

same as

```
int total =10;
if (total > 5)
  total = total + 1;
else
  total = total * 2;
```

The `for` statement

- Essentially the same as in C:

```
for (int i = 0; i < 3; i++) {
    System.out.println("i = " + i);
}
```

Generates:

```
i = 0
i = 1
i = 2
```

The `int` is declared <u>and</u> initialised in the `for` statement.

More about the `for` statement

Backwards

```
for (int n = 10; n >= -6; n--)
```

Empty

```
for (int n = 0; n <= -6; n++)
```

Nested

```
for (int i = 0; i <= 10; i++)
  for (int j = 0; j <= 78; j++)
```

Endless

```
for (int n = 0; ; n++)
for ( ; ; )
```

Unfinished use of **`break`** to pass control to the end of the loop

```
for (int n=0; n<=30; n++) {
  // other code
  if(n==10) break; // leaves the for loop when n=10
}
```

The `while` and `do-while` statements

```
int i = 0;
do {
   System.out.println("i = " + i);
   i++;
} while (i < 3);
```

OR

```
int i = 0;
while (i < 3) {
   System.out.println("i = " + i);
   i++;
}
```

Generates (for both examples):

```
i = 0
i = 1
i = 2
```

How can we write this code as a `for` loop?

Examples: Moving the i++ and resulting behaviour

```
int i = 0;
do {
   i++;
   System.out.println("i = " + i);
} while (i < 3);
```

Generates:

```
i = 1
i = 2
i = 3
```

OR

```
int i = 0;
while (i < 3) {
   i++;
   System.out.println("i = " + i);
}
```

Generates:

```
i = 1
i = 2
i = 3
```

```
int i = 3;
do {
   System.out.println("i = " + i);
   i++;
} while (i < 3);
```

Generates:

```
i = 3
```

OR

```
int i = 3;
while (i < 3) {
   System.out.println("i = " + i);
   i++;
}
```

Generates:

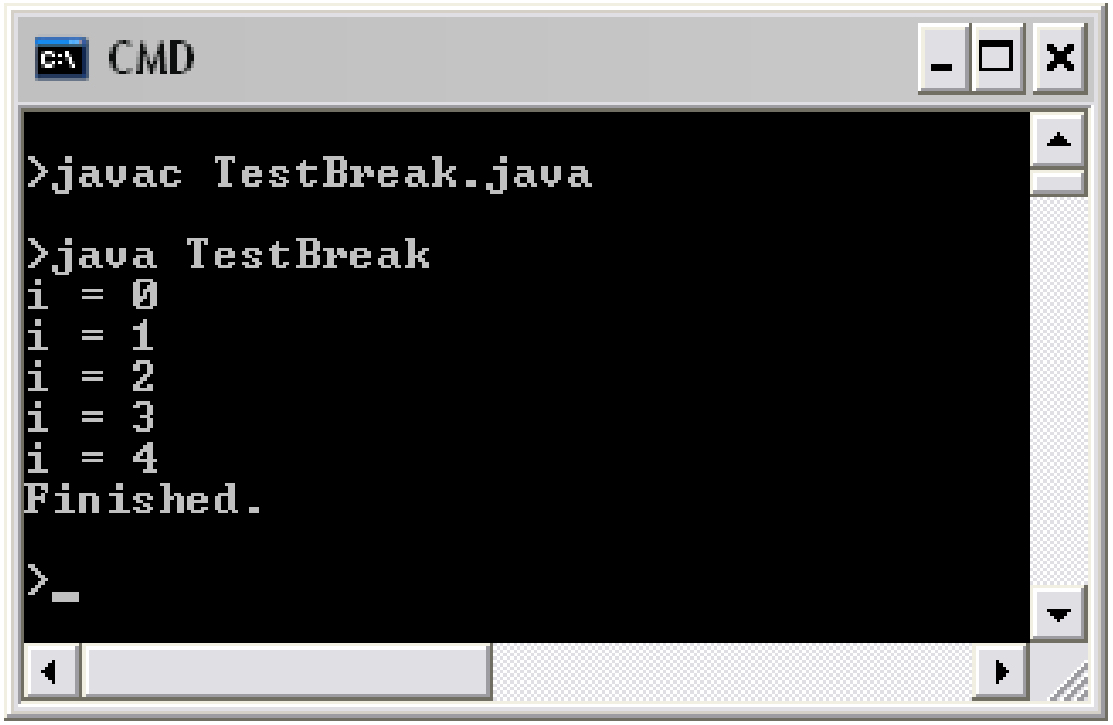
nothing

Queen Mary University of London

The break statement

```
class TestBreak {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 5) break;
      System.out.println("i = " + i);
    }
    System.out.println("Finished.");
  }
}
```

Output:



Quitting the loop …

The continue statement

```
class TestContinue {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 5) continue;
      System.out.println("i = " + i);
    }
    System.out.println("Finished.....");
  }
}
```

Output:

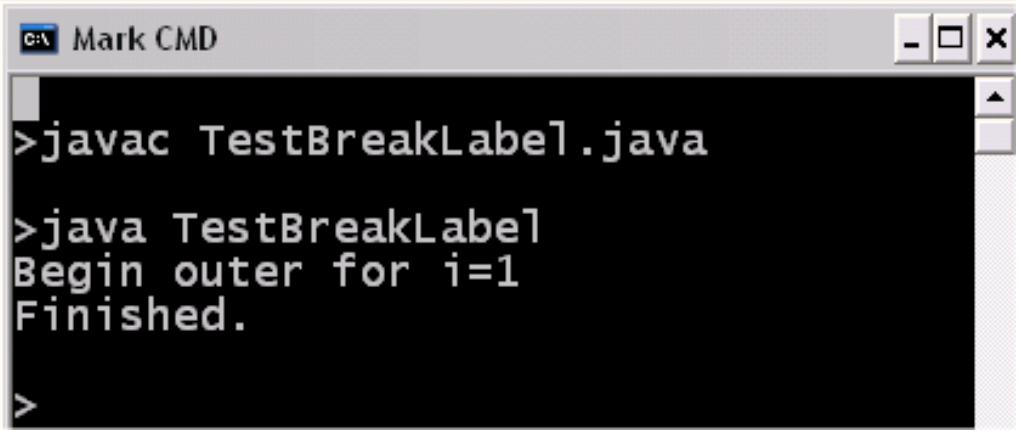
Skipping the current iteration …

Using labelled statement blocks with break (1/2)

```
class TestBreakLabel {
  public static void main(String[] args) {
    outer:
      for (int i = 1; i < 5; i++) {
        System.out.println("Begin outer for i=" + i);
        inner:
          for (int j = 1; j < 5; j++) {
            if (j == i) break outer;
              System.out.println("  inner: i=" + i + " j= " + j);
          }
          System.out.println("End outer for i=" + i);
      }
      System.out.println("Finished.");
  }
}
```

This is just a line marker, NOT a block.

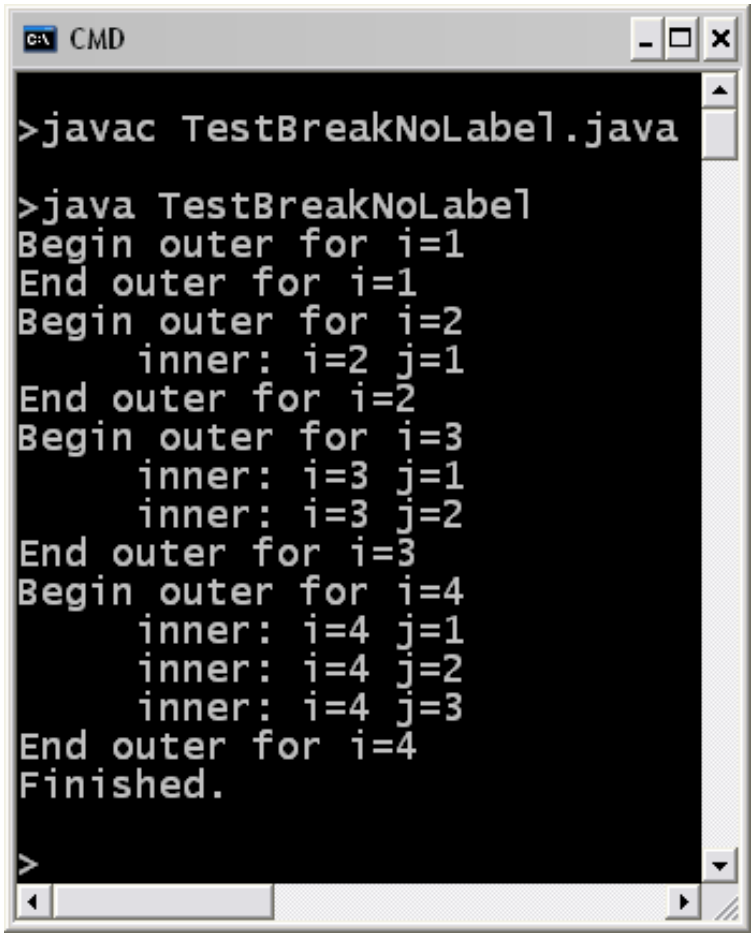
Use carefully!



Queen Mary University of London

Using labelled statement blocks with break (2/2)

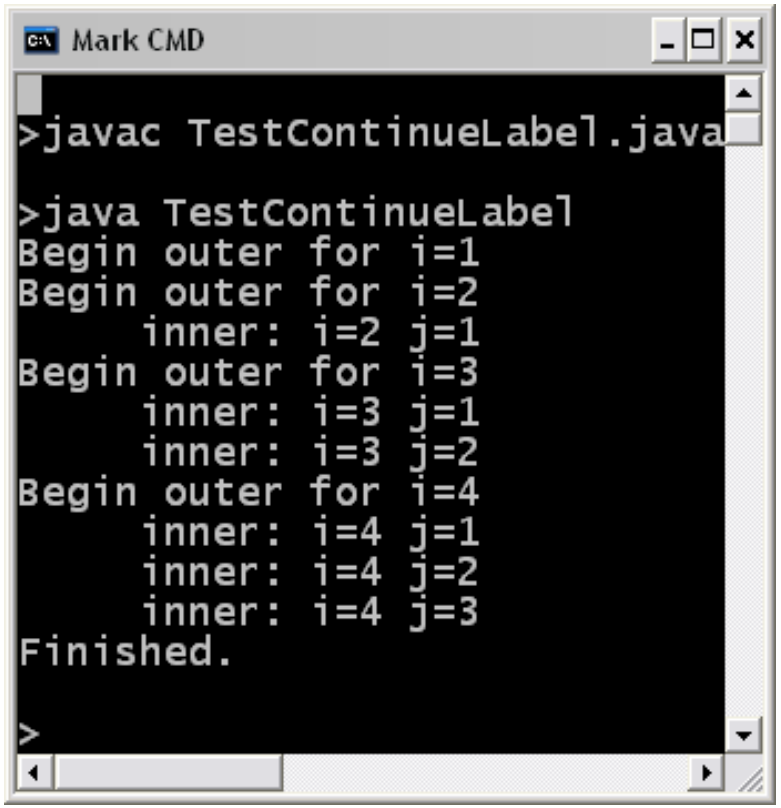
```
class TestBreakNoLabel {
  public static void main(String[] args) {
    outer:
      for (int i =1; i < 5; i++) {
        System.out.println("Begin outer for i=" + i);
        inner:
          for (int j = 1; j < 5; j++) {
            if (j == i) break;
              System.out.println(
                   "   inner: i=" +
                   i + " j=" + j );
          }
          System.out.println(
               "End outer for i=" + i);
      }
      System.out.println("Finished.");
  }
}
```



Use carefully!

Using labelled statement blocks with continue (1/2)

```
class TestContinueLabel {
  public static void main(String[] args) {
    outer:
      for (int i = 1; i < 5; i++) {
        System.out.println("Begin outer for i="+i);
        inner:
          for (int j = 1; j < 5; j++) {
            if (j == i) continue outer;
              System.out.println(
                      "   inner: i=" +
                      i + " j=" + j);
          }
          System.out.println(
               "End outer for i=" + i);
      }
      System.out.println("Finished.");
  }
}
```



Use carefully!

Using labelled statement blocks with continue (2/2)

```
class TestContinueNoLabel {
  public static void main(String[] args) {
    outer:
      for (int i = 1; i < 5; i++) {
        System.out.println(
              "Begin outer for i=" + i);
        inner:
          for (int j = 1; j < 5; j++) {
            if (j == i) continue;
              System.out.println(
                      "   inner: i=" +
                      i + " j="+j );
          }
          System.out.println(
               "End outer for i=" + i);
      }
      System.out.println("Finished.");
  }
}
```

```
CMD
>java TestContinueNoLabel
Begin outer for i=1
     inner: i=1 j=2
     inner: i=1 j=3
     inner: i=1 j=4
End outer for i=1
Begin outer for i=2
     inner: i=2 j=1
     inner: i=2 j=3
     inner: i=2 j=4
End outer for i=2
Begin outer for i=3
     inner: i=3 j=1
     inner: i=3 j=2
     inner: i=3 j=4
End outer for i=3
Begin outer for i=4
     inner: i=4 j=1
     inner: i=4 j=2
     inner: i=4 j=3
End outer for i=4
Finished.
```

Use carefully!

… and things for you to try out!

Exercises

5 minutes

1. Write Java statement(s) to determine if the value of an integer variable **num** is even or odd.
2. What is the output of the program below?

```
public class Test {
  public static void main(String[] args) {
    int i = 5;
    while (i >= 1) {
      int num = 1;
      for (int j = i; j <= i; j++) {
        System.out.print(num + "xxx");
        num *= 2;
      }
      System.out.println();
      i--;
    }
  }
}
```