Object-Oriented Programming
Variables, Conditionals, Loops and Arrays

Ewan Klein

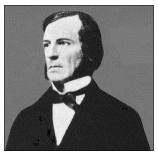
School of Informatics

Inf1 :: 2009/10

Equality

$$x == 3$$

- This is a boolean statement;
- i.e., either true or false.
- Used as a test — what is the current state of the computation?



Assignment & Equality

- `int num;`
  declare num as a variable.
- `num = 5;`
  assign 5 as the value of num.
- `num = num * 2;`
  multiply the current value of num by 2 and make this the new value of num.
- `if (num == 4) { //do something}`
  Test to see if the current value of num is 4.

The Tower Example

```
public class BuildTower {
    public static void main(String[] args) {
        int towerHeight = 0;
        int targetHeight = 10;

        while (towerHeight < targetHeight) {
            towerHeight = towerHeight + 1;
            System.out.print(towerHeight + " ");
        }
    }
}
```

Output

```
1 2 3 4 5 6 7 8 9 10
```


The Unary Increment Operator: ++

```
public class BuildTower {
    public static void main(String[] args) {
        int towerHeight = 0;
        int targetHeight = 10;

        while (towerHeight < targetHeight) {
            \\ same as towerHeight = towerHeight + 1;
            towerHeight++;
            System.out.print(towerHeight + " ");
        }
    }
}
```


Compound Assignment: +=

```
public class BuildTower {
    public static void main(String[] args) {
        int towerHeight = 0;
        int targetHeight = 10;

        while (towerHeight < targetHeight) {
            \\ same as towerHeight = towerHeight + 1;
            towerHeight += 1;
            System.out.print(towerHeight + " ");
        }
    }
}
```

NB

Could also have
`towerHeight += 3;`

One Thing After Another

```
String yrName = "Juan";
yrName = yrName + " Jr";
String myName = yrName;
```

Value of myName is "Juan Jr".

```
String yrName = "Juan";
String myName = yrName;
yrName = yrName + " Jr";
```

Value of myName is "Juan".

NB

The operator + is overloaded — it means addition on numbers and concatenation on strings.

Testing Boolean Conditions

```
boolean coffeeCupNotEmpty = True;

while (coffeeCupNotEmpty) {
   keepDrinking();
}
```

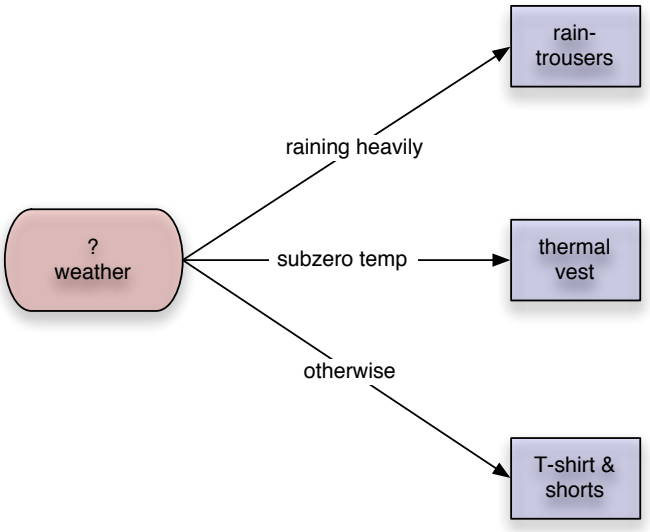
```
int moneyInPocket = 200;

while (moneyInPocket > 50) {
   // spend some money
   moneyInPocket =  moneyInPocket - 5;
}
```


Things to Test

- `coffeeCupNotEmpty` is a variable of type `boolean` — the only values it can take are `true` and `false`.
- `moneyInPocket > 50` is a boolean expression — it evaluates to `true` or `false`.
- >, < and == are examples of comparison operators.
- `while` (*boolean*) `{ ... }` — *boolean* expresses a condition to be tested.

Branching

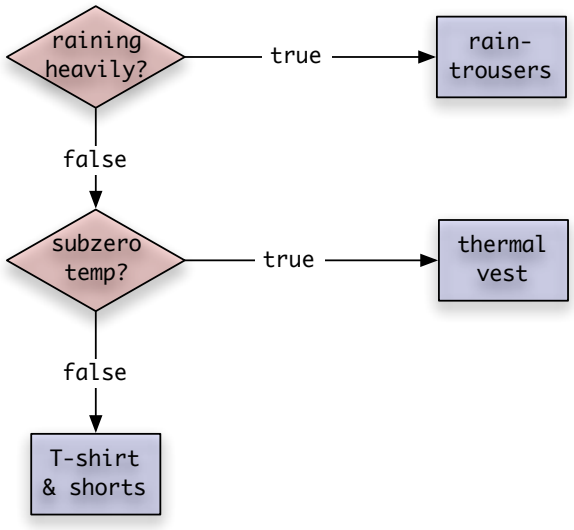


Edinburgh Weather Example

- We already saw a conditional expression:
  `if (num == 4) { ...}`

```
if (rainingHeavily) {
   clothes = "Rain-trousers";
} else if (temp < 0) {
   clothes = "Thermal vest";
} else {
   clothes = "T-shirt and shorts";
}
```

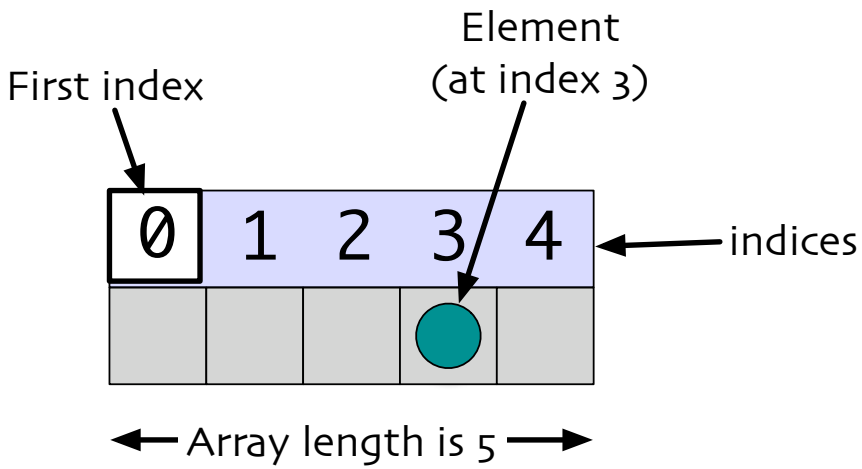
A Better Branching Diagram



Arrays, 1

- A container object — holds fixed number of values of a single type.
- Length of array is set at creation time, cannot be changed.
- Items in an array called elements.
- Elements are accessed by numerical index.

Arrays, 2



Arrays, 3

- Type of array: *type*[], where *type* is data type of elements.

Declare, then initialise

```
int[] ratings; // declare array of ints
ratings = new int[5]; // allocate memory
```

Declare and initialise

```
int[] ratings = new int[5];
```

Initialise some elements

```
ratings[0] = 33;
ratings[1] = 51;
...
```

Arrays, 4

Accessing elements

```
ratings[0]; // evaluates to 33
ratings[1]; // evaluates to 51
...
```

Shortcut Syntax

```
int[] ratings = new {33, 51, 7, 89, 14};
// array of length 5
```

Get the length

```
ratings.length; // evaluates to 5
```


Arrays, 5

Create a Dog array

```
Dog[] myDogs = new Dog[7];
```

Elements?

```
myDogs[0]; // won't work
```

Create Dog objects as elements

```
myDogs[0] = new Dog(); // myDogs[0] is a reference variable
myDogs[1] = new Dog();
```

Do something with a Dog

```
myDogs[0].run();
```


for Loops (HFJ pp114–115)

- `for` (*initialization*; *boolean test*; *iteration*) `{ ... }`

Example for loop

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- Create variable `i` and set it to `0`.
- Repeat loop while `i` is less than `10`.
- At the end of each loop iteration, increment `i` by 1.

Loop through an array

```
for (int i = 0; i < ratings.length; i++) {
    if (ratings[i] < 20) {
        System.out.println(ratings[i]);
    } // close if
} // close for loop
```


Summary

- Imperative program is a sequence of 'statements' or commands.
- Each statement can change the current state of the program (i.e., values of variables).
- The Java interpreter generally steps through the sequence of commands from top to bottom.
- Control structures (`while`, `if`) change the order of execution — code is executed under certain conditions.
- `while` (*condition*) { *statements* } will repeatedly evaluate *statements* as long as *condition* holds true.
- `if` (*condition*) { *statements* } will only evaluate *statements* if *condition* holds true.

Bullet Points, 1

- OOP helps you to program in a modular way
- All Java code is located in a class.
- Objects are instances of a class.
- Objects have state, specified by instance variables.
- Two kinds of variable: primitive and reference.
- NB instance variables is more about use; can be both primitive and reference.
- A reference variable contains information about how to get to an object, not the object itself.
- Objects have behaviour, specified by methods.
- Use the dot operator to call a method on a reference variable (e.g., `dog.bark()`).
- An array is an object whose elements are variables of some type, either primitive or reference.
- Length of array is fixed at creation time.
- Elements are accessed by index (counting from 0).

Reading, Labs, etc

- Chapters 2, 3 of HFJ.
- Chapter 4 might also help with Lab Exercises.
- Last Lab Exercise (2.5) is optional.
- Next lecture: Tuesday 27th Jan.