

# Inf1 OOP: Java Reference Sheet

Ewan Klein & Daniel Powell — December 22, 2011

---

## 1 Defining a Class

The following is a schematic class definition, illustrating the main structure.

```
1 public class ClassName {
2
3     private int varint;
4     private boolean varbool = True;
5
6     public void methodName1() {
7         ...
8     }
9
10    public boolean methodName2( int parameterName ) {
11        ...
12        return ( true_or_false_expression );
13    }
14
15    public static void main( String[] args ) {
16        ...
17        ClassName varName = new ClassName();
18        varName.methodName1();
19
20        boolean result = varName.methodName2( number );
21    }
22
23 }
```

- (1)–(23) This is the main definition of a class. The `ClassName` should start with a capital and must match the filename (e.g., `ClassName.java`).
- (3)–(4) This is where you declare and initialize instance variables, i.e., variables that hold the state of every instance of your class. `varint` and `varbool` both have primitive types.
- (6)–(8) This defines a method `methodName1` within a class. This method has no parameters and `void` means it returns nothing.
- (10)–(13) This defines another method `methodName2` within a class. It returns a boolean (`true` or `false` value). See below for more types.
- (15)–(21) The `main()` method is run first when the program is executed. Not every class contains a `main()` method.
  - (17) Here, we declare a reference variable `varName` with type `ClassName` and link it to a new instance of the class `ClassName()`.
  - (18) We call a method `methodName1` of the class `ClassName`.

(20) We call the method `methodName2` of the class, passing it an argument `number`, and store the return value `methodName2` in the variable `result`.

Here is simple, concrete, example of a class:

```
1 public class TestRun {
2     public static final void main(String[] args) {
3         System.out.println("Hello World.");
4     }
5 }
```

## Primitive Variables, Return Values

- `boolean` — true or false value
- `int`, `long` — whole numbers; `int` will normally suffice
- `float`, `double` — decimal numbers; `double` is the default.
- `void` — returns nothing!

## 2 Operators

Here's a list of frequently-used operators.

<b>postfix</b>	<i>expr++ expr--</i>
<b>arithmetic</b>	<i>* / % + -</i>
<b>equality</b>	<i>== !=</i>
<b>comparison</b>	<i>&gt; &gt;= &lt; &lt;=</i>
<b>logical AND</b>	<i>&amp;&amp;</i>
<b>logical OR</b>	<i>  </i>
<b>logical NOT</b>	<i>!</i>
<b>assignment</b>	<i>= += -=</i>

## 3 Control

**if statement**

```
1 if ( true_or_false_expression ) {
2     ...
3 } else if ( true_or_false_expression ) {
4     ...
5 } else {
6     ...
7 }
```

This is an `if` statement, controlled by a Boolean expression inside the parentheses. The `else if` and `else` parts of the statement in (3)–(7) are optional.

## while loop

```
1 while( true_or_false_expression )  
2   { ... }
```

A simple `while` loop: while the boolean expression in parentheses is true, the code in the block will be executed.

## for loop

```
1 int[] anArray = new int[10];  
2 for (int i = 0; i < anArray.length; i++)  
3   { ... }
```

- (1) Create a new array of integers of length 10, and link it to the variable `anArray`. The first element of `anArray` is accessed by `anArray[0]`, and its last element is accessed by `anArray[9]`.
- (2) Use the “loop variable” `i` to loop over each index in `anArray`, incrementing `i` each time through the loop. We break out of the loop as soon as `i < anArray.length` (i.e., `i < 10`) becomes false.

The next example shows how to print out every other element of an array.

```
1 int[] anArray = {5, 3, 6, 9, 1};  
2 for (int i = 0; i < anArray.length; i += 2) {  
3   System.out.println(anArray[i]);  
4   }
```

- (1) Directly initialize an array of 5 integers.
- (2) Incrementing `i` by 2 each time through the loop.
- (3) Print out the value of `anArray[i]` each time through the loop. This will produce

```
5  
6  
1
```

## ‘for each’ loop

```
1 String[] pasta = { "spaghetti", "penne", "tagliatelle"};  
2  
3 for (String p : pasta) {  
4   System.out.println(p);  
5 }
```

If you don’t need to use the array index, ‘for each’ loops are the preferred way of looping over any array-like object.

## 4 Collections and Generic Types

```
1 import java.util.*
2 ...
3 ArrayList<String> mylist = new ArrayList<String>();
4 HashMap<String, Integer> myMap = new HashMap<String, Integer>();
```

The types in the angle brackets ('generic' types) have to be reference types. Every primitive type has a corresponding reference type.

- (1) Import collections such as `ArrayList` and `HashMap` from the `java.util` package.
- (3) Declare and initialize an `ArrayList` to hold objects of type `String`.
- (4) Declare and initialize a `HashMap` to associate objects of type `String` with objects of type `Integer`.

### Some Methods of ArrayList

<code>add(Object element)</code>	Add the object <code>element</code> to the list
<code>remove(int index)</code>	Remove the object at <code>index</code>
<code>remove(Object element)</code>	Remove the object <code>element</code>
<code>contains(Object element)</code>	Returns <code>true</code> if the list contains <code>element</code>
<code>isEmpty()</code>	Returns <code>true</code> if the list has no elements
<code>indexOf(Object element)</code>	Returns the index of <code>element</code> or <code>-1</code>
<code>size()</code>	Returns the number of elements in the list
<code>get(int index)</code>	Returns the object at <code>index</code>

### Some Methods of HashMap

<code>put(key, value)</code>	Map <code>key</code> to <code>value</code>
<code>get(key)</code>	Get the value associated with <code>key</code>
<code>remove(key)</code>	Remove the mapping that has <code>key</code> as key
<code>containsKey(key)</code>	Returns <code>true</code> if the map contains <code>key</code> as a key

## 5 Math static methods

<code>Math.random()</code>	Returns a <code>double</code> in the interval <code>[0.0, 1.0)</code>
<code>Math.abs()</code>	Returns a <code>double</code> that is the absolute value of the argument
<code>Math.round()</code>	Given a <code>float</code> or <code>double</code> argument, returns an <code>int</code> or a <code>long</code> respectively, rounded to the nearest integer value
<code>Math.min()</code>	Returns the minimum of two arguments
<code>Math.max()</code>	Returns the maximum of two arguments

## 6 Constructors, Abstract Classes and Interfaces

### Constructors

A constructor contains the code that runs when you call `new` on a class type. A constructor looks like a method (including a parameter list), except the name is the same as the class and there's no return type.

```
1 public class Cell {
2     private int location;
3     public Cell(int loc) {
4         location = loc;
5     }
6 }
7
8 Cell c = new Cell(35);
```

```
1 public class PlantCell extends Cell {
2
3     public PlantCell(int loc) {
4         super(loc);
5     }
6 }
```

- (1) `PlantCell` is a subclass of `Cell`.
- (4) The constructor for `PlantCell` calls the constructor for its superclass, namely `Cell`, with the argument given by `loc`.

### Abstract Classes

```
1 public abstract class Canine extends Animal {
2     public abstract String roam();
3
4     public String sleep() {
5         return "Sleeping: Prrrr!";
6     }
7 }
```

- (1)–(7) Declare an abstract class — it must be extended by a concrete subclass.
- (2) Declare an abstract method — this must be overridden by a method in the subclass.
- (4)–(6) This is an implemented method. It can be inherited or overridden in the subclass.
- (3) Declare and initialize a `HashMap` to associate objects of type `String` with objects of type `Integer`.

## Interfaces

```
1 public interface MyInterface {  
2  
3     boolean method1(Valuation val);  
4     String method2();  
5 }
```

(1)–(5) Declare an interface.

(3), (4) Declare two methods — these are both implicitly abstract and public. Every concrete class that implements the interface *must* implement both of these methods.

## Implements

```
1 public class SubClass extends SuperClass implements Intface1, Intface2 {  
2     ...  
3 }
```

(1) Declare a class which extends `SuperClass` and implements two interfaces.

## 7 String Formatting

```
1 int age = 18  
2 System.out.printf("The student named '%s' is aged %s.", "Peter", age);  
3  
4 String str = String.format("The student named '%s' is aged %s.", "Peter", age);  
5 System.out.println(str);
```

(2) Prints out "The student named 'Peter' is aged 18."

(4) Prints out "The student named 'Peter' is aged 18."

## Escape sequences

newline	\n
tab	\t
backslash	\\