
Programming overview

Basic Java

- A Java program consists of:
 - One or more *classes*
 - A class contains one or more *methods*
 - A method contains *program statements*
 - Each class in a separate file
 - MyClass defined in MyClass.java
 - Compiled to java bytecode
 - To define MyClass.class
-

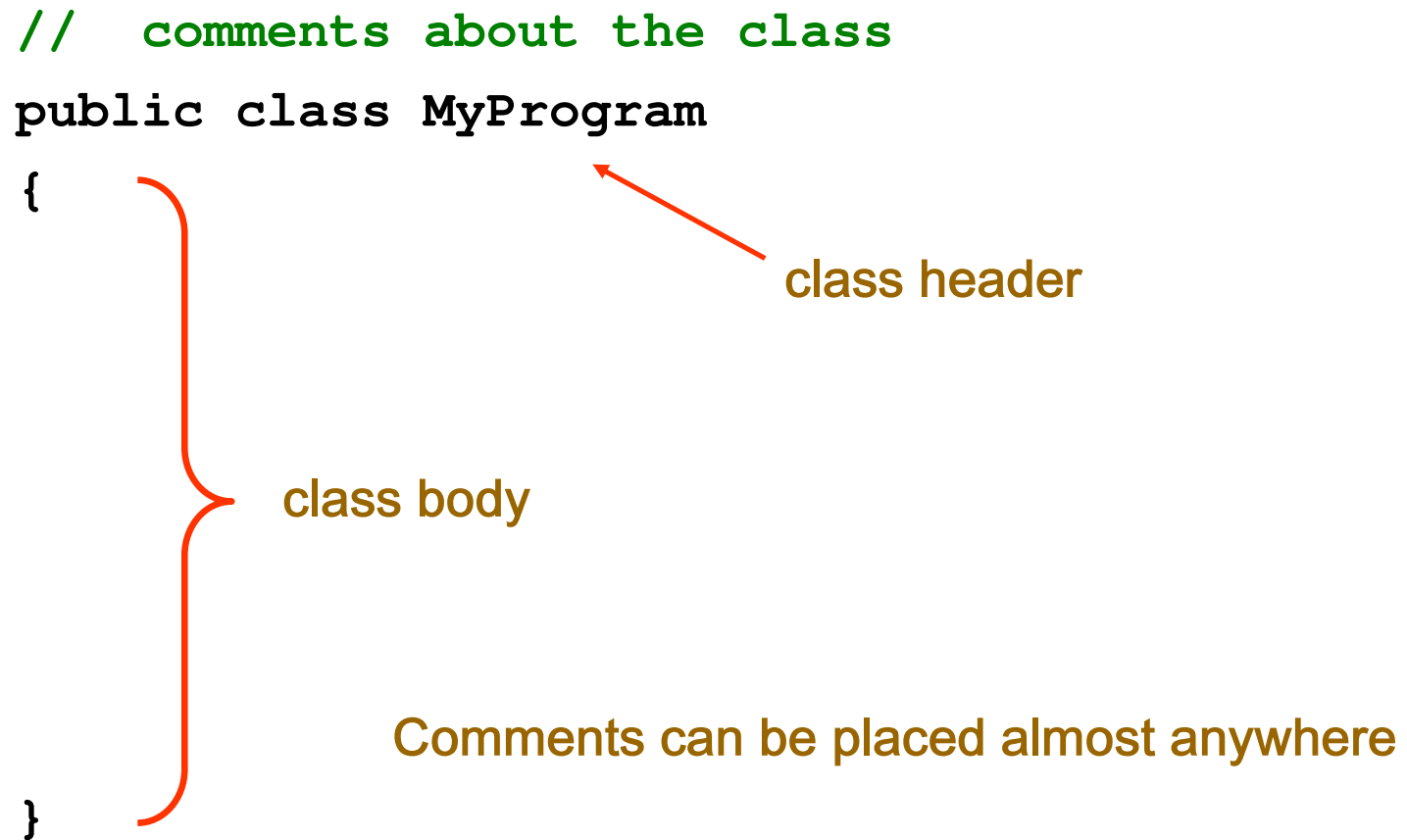
Basic Java

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

class header



class body

Comments can be placed almost anywhere

```
}
```

Basic Java

```
// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
    }
}
```



method header

method body

Hello World

```
// HelloWorld.java
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Program Structure

- Typical Java program consists of
 - User written classes
 - Java Application Programming Interface (API) classes
 - Java application
 - Has one class with a *main* method
 - Java program basic elements:
 - Packages
 - Classes
 - Data fields
 - Methods
-

Packages

- Provide a mechanism for grouping related classes
 - `package` statement
 - Indicates a class is part of a package
 - Java assumes all classes in a particular package are contained in same directory
 - Java API consists of many predefined packages
-

Packages

- `import` statement
 - Allows you to use classes contained in other packages
 - Package `java.lang` is implicitly imported to all Java code
-

	File SimpleSphere.java
1. Indicates SimpleSphere is part of a package ---->	package MyPackage;
2. Indicates class Math is used by SimpleSphere -->	import java.lang.Math;
3. Begins class SimpleSphere ----->	public class SimpleSphere {
4. Declares a private data field radius ----->	private double radius;
5. Declares a constant ----->	public static final double DEFAULT_RADIUS = 1.0;
6. A default constructor ----->	public SimpleSphere() {
7. Assignment statement ----->	radius = DEFAULT_RADIUS;
	} // end default constructor
8. A second constructor ----->	public SimpleSphere(double r) {
9. Assignment statement ----->	radius = r;
	} // end constructor
10. Begins method getRadius ----->	public double getRadius() {
11. Returns data field radius ----->	return radius;
	} // end getRadius
12. Begins method getVolume ----->	public double getVolume() {
13. A comment ----->	// Computes the volume of the sphere.
14. Declares and assigns a local variable ----->	double radiusCubed = radius * radius * radius;
15. Returns result of computation ----->	return 4 * Math.PI * radiusCubed / 3;
	} // end getVolume
16. Ends class SimpleSphere ----->	} // end SimpleSphere
	File TestClass.java
17. Indicates TestClass is part of a package ----->	package MyPackage;
18. Begins class TestClass ----->	public class TestClass {
19. Begins method main ----->	static public void main(String[] args) {
20. Declares reference ball ----->	SimpleSphere ball;
21. Creates a SimpleSphere object ----->	ball = new SimpleSphere(19.1);
22. Outputs results ----->	System.out.println("The volume of a sphere of radius "
23. Continuation of output string ----->	+ ball.getRadius() + " inches is "
24. Continuation of output string ----->	+ (float)ball.getVolume()
	+ "cubic inches\n");
	} //end main
25. Ends class TestClass ----->	} // end TestClass

Figure 1-1

A simple Java Program

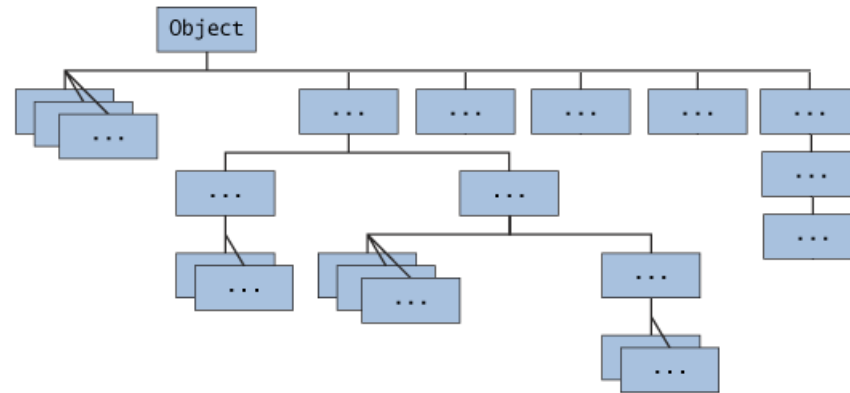
Classes

- An object in Java is an instance of a class
 - Class definition includes
 - ❑ Optional subclassing modifier
 - ❑ Optional access modifier
 - ❑ Keyword `class`
 - ❑ Optional `extends` clause
 - ❑ Optional `implements` clause
 - ❑ Class body
-

Inheritance

- A class can inherit the members of another class.
 - A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*).
 - The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).
 - Excepting *Object* which is a system defined class and has no superclass, every class has **one and only one direct** superclass (single inheritance).
 - In the absence of any other explicit superclass, every class is implicitly a subclass of *Object*.
-

Class hierarchy



All Classes in the Java Platform are Descendants of Object

Inheritance

- A subclass inherits all the public and protected *members* (data, methods, and nested classes) from its superclass.
 - Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the constructor of a subclass using the *super* keyword.
 - A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.
-

Inheritance example

```
public class Bicycle {  
  
    // the Bicycle class has three fields  
    public int cadence;  
    public int gear;  
    public int speed;  
  
    // the Bicycle class has one constructor  
    public Bicycle(int startCadence, int startSpeed, int startGear) {  
        gear = startGear;  
        cadence = startCadence;  
        speed = startSpeed;  
    }  
  
    // the Bicycle class has four methods  
    public void setCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    public void setGear(int newValue) {  
        gear = newValue;  
    }  
  
    public void applyBrake(int decrement) {  
        speed -= decrement;  
    }  
  
    public void speedUp(int increment) {  
        speed += increment;  
    }  
}
```

```
public class MountainBike extends Bicycle {  
  
    // the MountainBike subclass adds one field  
    public int seatHeight;  
  
    // the MountainBike subclass has one constructor  
    public MountainBike(int startHeight, int startCadence,  
        int startSpeed, int startGear) {  
        //calling the constructor of the superclass.  
        super(startCadence, startSpeed, startGear);  
  
        seatHeight = startHeight;  
    }  
  
    // the MountainBike subclass adds one method  
    public void setHeight(int newValue) {  
        seatHeight = newValue;  
    }  
    //overriding a method from the super-class  
    public void applyBrake(int increment){  
        speed += 2*increment;  
    }  
}
```

What You Can Do in a Subclass

- The inherited fields (data member) can be used directly, just like any other fields.
 - You can declare a field in the subclass with the same name as the one in the superclass, thus *hiding* it (not recommended).
 - You can declare new fields in the subclass that are not in the superclass.
 - The inherited methods can be used directly as they are.
 - You can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus *overriding* it.
 - You can declare new methods in the subclass that are not in the superclass.
 - You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword `super`.
-

Nested Classes

- The Java programming language allows you to define a class within another class.
- Such a class is called a *nested class*.

```
class OuterClass {  
    ...  
    class NestedClass {  
        ...  
    }  
}
```

- A nested class is a member of its enclosing class and, as such, has access to other members of the enclosing class, even if they are declared private.
-

Why Use Nested Classes?

- It is a way of logically grouping classes that are only used in one place.
 - It increases encapsulation.
 - Nested classes can lead to more readable and maintainable code.
-

Classes definition components

Component	Syntax	Description
Subclassing modifier (use only one)	abstract	Class must be extended to be useful.
	final	Class cannot be extended.
Access modifiers	public	Class is available outside of package.
	no access modifier	Class is available only within package.
Keyword class	class <i>class-name</i>	Class should be contained in a file called <i>class-name.java</i> .
extends clause	extends <i>superclass-name</i>	Indicates that this class is a subclass of the class <i>superclass-name</i> in the <i>extends</i> clause.
implements clause	implements <i>interface-list</i>	Indicates the interfaces that this class implements. The <i>interface-list</i> is a comma-separated list of interface names.
Class body	Enclosed in braces	Contains data fields and methods for the class.

Figure 1-2

Components of a class

Data Fields

- Class members that are either variables or constants
 - Data field declarations can contain
 - Access modifiers
 - Use modifiers
 - Modules
-

Data Fields

Type of modifier	Keyword	Description
Access modifier (use only one)	public	Data field is available everywhere (when the class is also declared <i>public</i>).
	private	Data field is available only within the class.
	protected	Data field is available within the class, available in subclasses, and available to classes within the same package.
	No access modifier	Data field is available within the class and within the package.
Use modifiers (all can be used at once)	static	Indicates that only one such data field is available for all instances of this class. Without this modifier, each instance has its own copy of a data field.
	final	The value provided for the data field cannot be modified (a constant).
	transient	The data field is not part of the persistent state of the object.
	volatile	The value provided for the data field can be accessed by multiple threads of control. Java ensures that the freshest copy of the data field is always used.

Figure 1-3

Modifiers used in data field declarations

Methods

- Used to implement operations
 - Should perform one well-defined task
 - Method modifiers
 - Access modifiers and use modifiers
 - Valued method
 - Returns a value
 - Body must contain **return** `expression`;
-

Method Modifiers

Type of modifier	Keyword	Description
Access modifier (use only one)	public	Method is available everywhere (when the class is also declared as <i>public</i>).
	private	Method is available only within the class (cannot be declared <i>abstract</i>).
	protected	Method is available within the class, available in subclasses, and available to classes within the same package.
	No access modifier	Method is available within the class and to classes within the package.
Use modifiers (all can be used at once)	static	Indicates that only one such method is available for all instances of this class. Since a <i>static</i> method is shared by all instances, the method can refer only to data fields that are also declared <i>static</i> and shared by all instances.
	final	The method cannot be overridden in a subclass.
	abstract	The method must be overridden in a subclass.
	native	The body of the method is not written in Java but in some other programming language.
	synchronized	The method can be run by only one thread of control at a time.

Figure 1-4

Modifiers used in a method declaration

How to Access Members of an Object

- `new` operator

- Creates an object or instance of a class
- The newly created object has brand new members (data and methods) except for the *static members*.
- The static members are common between all the objects of a class.

- Data fields and methods declared *public*

- Name the object, followed by a period, followed by member name

- Members declared *static*

- Use the class or object name, followed by a period, followed by member name
-

Interface.

- In its most common form, an interface is a group of related methods with empty bodies.
- To implement this interface, you need to define a class that implements all the methods in the interface class.

```
interface BicycleInterface {  
  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
  
}
```

```
public class Bicycle implements BicycleInterface{  
  
    // the Bicycle class has three fields  
    public int cadence;  
    public int gear;  
    public int speed;  
  
    // the Bicycle class has one constructor  
    public Bicycle(int startCadence, int startSpeed, int startGear) {  
        gear = startGear;  
        cadence = startCadence;  
        speed = startSpeed;  
    }  
  
    // the Bicycle class has four methods  
    public void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    public void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    public void applyBrake(int decrement) {  
        speed -= decrement;  
    }  
  
    public void speedUp(int increment) {  
        speed += increment;  
    }  
  
}
```

Variables

- Represents a memory location
 - Contains a value of *primitive* (for instance *int*) type or a *reference*. (it cannot be both)
 - **Primitive** types are types where the name of the variable evaluates to the value stored in the variable.
 - **Reference** types in Java are types where the name of the variable evaluates to the address of the location in memory where the object is stored.
-

Primitive Data Types

Category	Data Type	Wrapper Class
Boolean	<code>boolean</code>	<code>Boolean</code>
Character	<code>char</code>	<code>Character</code>
Integer	<code>byte</code>	<code>Byte</code>
	<code>short</code>	<code>Short</code>
	<code>int</code>	<code>Integer</code>
	<code>long</code>	<code>Long</code>
Floating point	<code>float</code>	<code>Float</code>
	<code>double</code>	<code>Double</code>

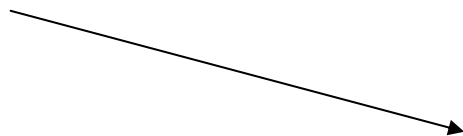
Figure 1-5

Primitive data types and corresponding wrapper classes

References

```
Public class aClass{  
    aClass(int value){m=value;    }  
    public int m;  
}  
aClass o = new aClass(225);
```

o



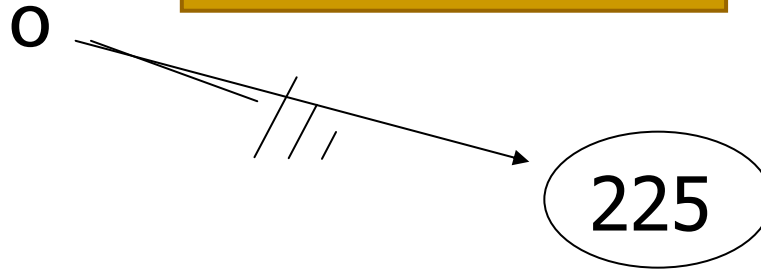
225

The variable o
is a reference
to the object

References

```
aClass o = new aClass(225);  
o = null;
```

Makes o not refer to
the object any more



References

```
aClass o = new aClass (225) ;  
o = null;
```

o

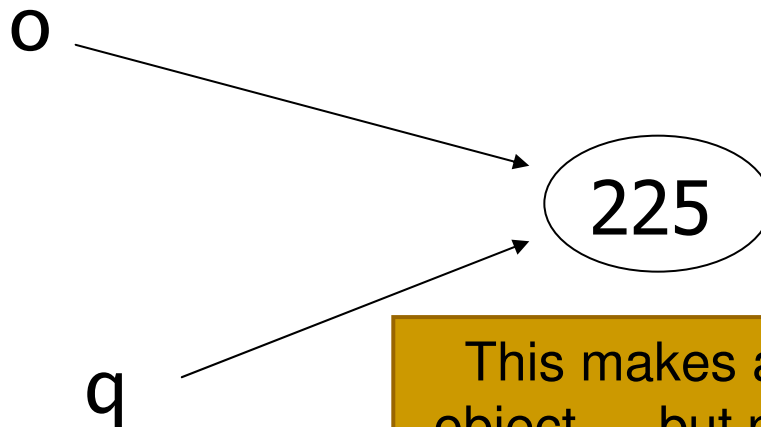
Makes o not refer to
the object any more

225

The object gets deleted
by Java's automated
garbage collection

References

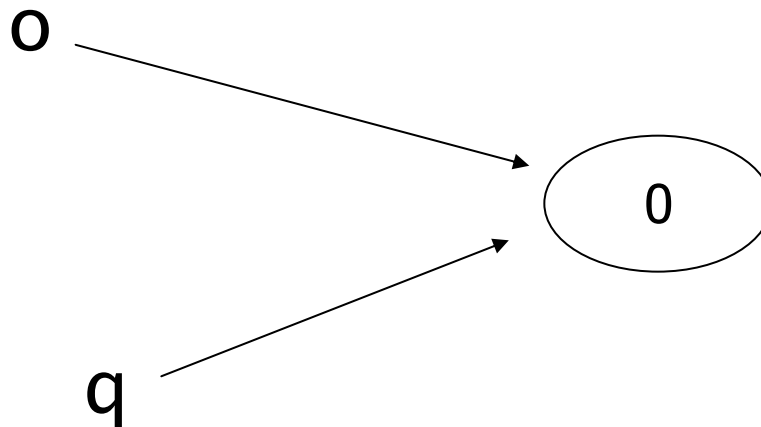
```
aClass o = new aClass(225);  
aClass q = o;
```



This makes another reference to the object --- but no new object is created!!

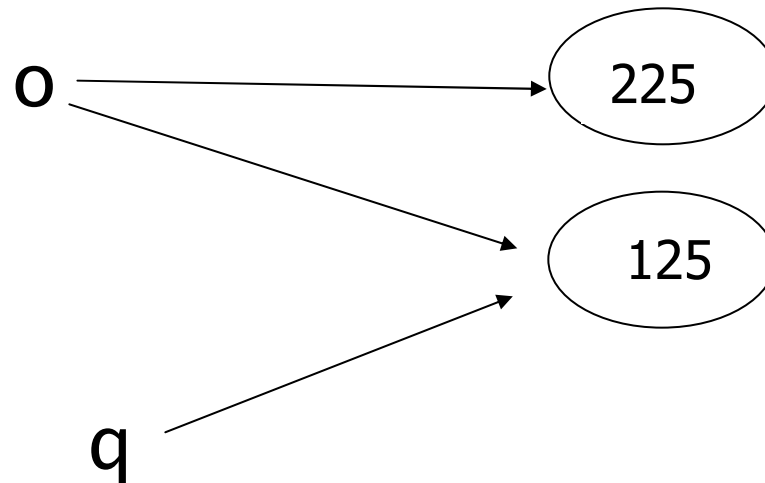
References

```
aClass o = new aClass(225);  
aClass q = o;  
q.m=0;
```



References

```
aClass o = new aClass (225) ;  
aClass q = new aClass (125) ;  
o=q;
```



The object gets deleted
by Java's automated
garbage collection

Parameter passing in Java - by reference or by value?

```
int x = 0;  
giveMeATen (x);  
System.out.println (x);  
[...]  
void giveMeATen (int y) {  
    y = 10;  
}
```

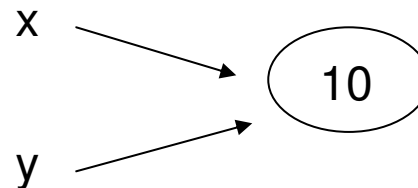
Output: 0

X: 0

Y: 10

```
aClass x = new aClass(0);  
giveMeATen (x);  
System.out.println (x.m);  
[...]  
void giveMeATen (aClass y) {  
    y.m = 10;  
}
```

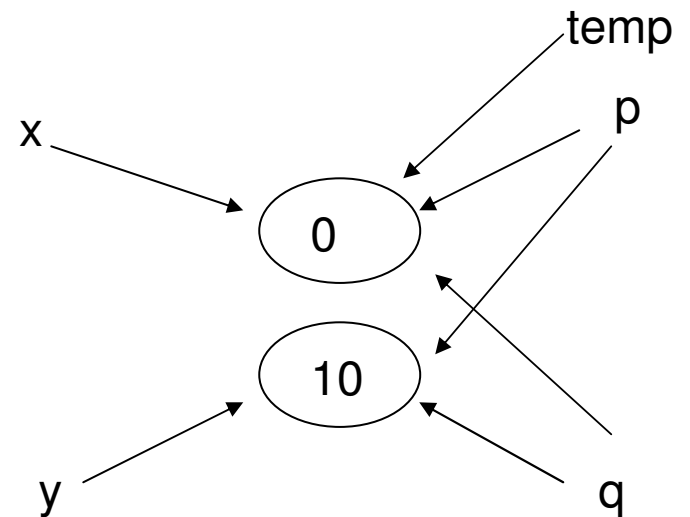
Output: 10



Myth: "Objects are passed by reference, primitives are passed by value"

```
aClass x = new aClass(0);  
aClass y = new aClass(10);  
swap (x, y);  
System.out.println (x.m+y.m);  
[...]  
void swap (aClass p, aClass q) {  
    aClass temp=p;  
    p = q; q = temp;  
}
```

Output: 0 10



Parameter passing in Java-The truth

- **Truth #1: Everything in Java is passed by value.**
- **Truth #2: The values of variables are always primitives or references, never objects.**



Arrays in Java

- To declare an array follow the type with (empty) []s
 - `int[] grade; //or`
 - `int grade[]; //both declare an int array`
 - In Java arrays are objects so must be created with the **new** keyword
 - To create an array of ten integers:
 - `int[] grade = new int[10];`
 - Note that the array size has to be specified, although it can be specified with a variable at run-time
-

Arrays in Java

- ❑ When the array is created memory is reserved for its contents
 - ❑ Initialization lists can be used to specify the initial values of an array, in which case the **new** operator is not used
 - `int[] grade = {87, 93, 35}; //array of 3 ints`
 - To find the length of an array use its `.length` variable
 - ❑ `int numGrades = grade.length; //note: not .length() !!`
-