

Java

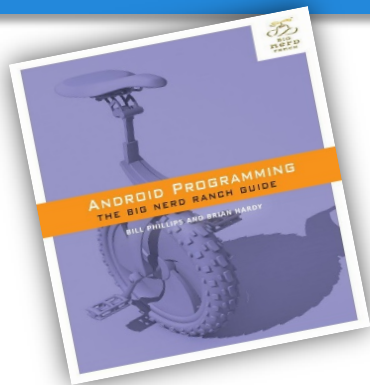
Contents

- Big Java
- Classes and objects
- Object-oriented programming (OOP)
- Interfaces
- Packages
- Google Java style

Contents

- **Big Java**
- Classes and objects
- Object-oriented programming (OOP)
- Interfaces
- Packages
- Google Java style

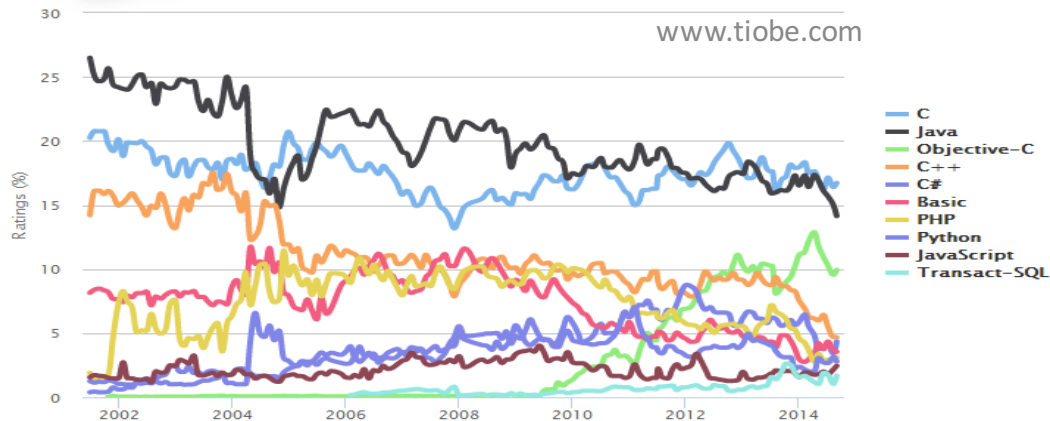
Big Java



*“...you need to be familiar with Java, including **classes and objects**, **interfaces**, **listeners**, **packages**, **inner classes**, **anonymous inner classes**, and **generic classes**.*

If these ideas do not ring a bell, you will be in the weeds by page 2...”

– Android Programming: The Big Nerd Ranch Guide



Contents

- Big Java
- **Classes and objects**
- Object-oriented programming (OOP)
- Interfaces
- Packages
- Google Java style

Classes and objects

Classes

A class is a template, blueprint, or contract that defines what an object's data fields and methods will be.

Class Name: Circle

Data Fields:
radius is _____

Methods:
getArea
getPerimeter
setRadius

Objects

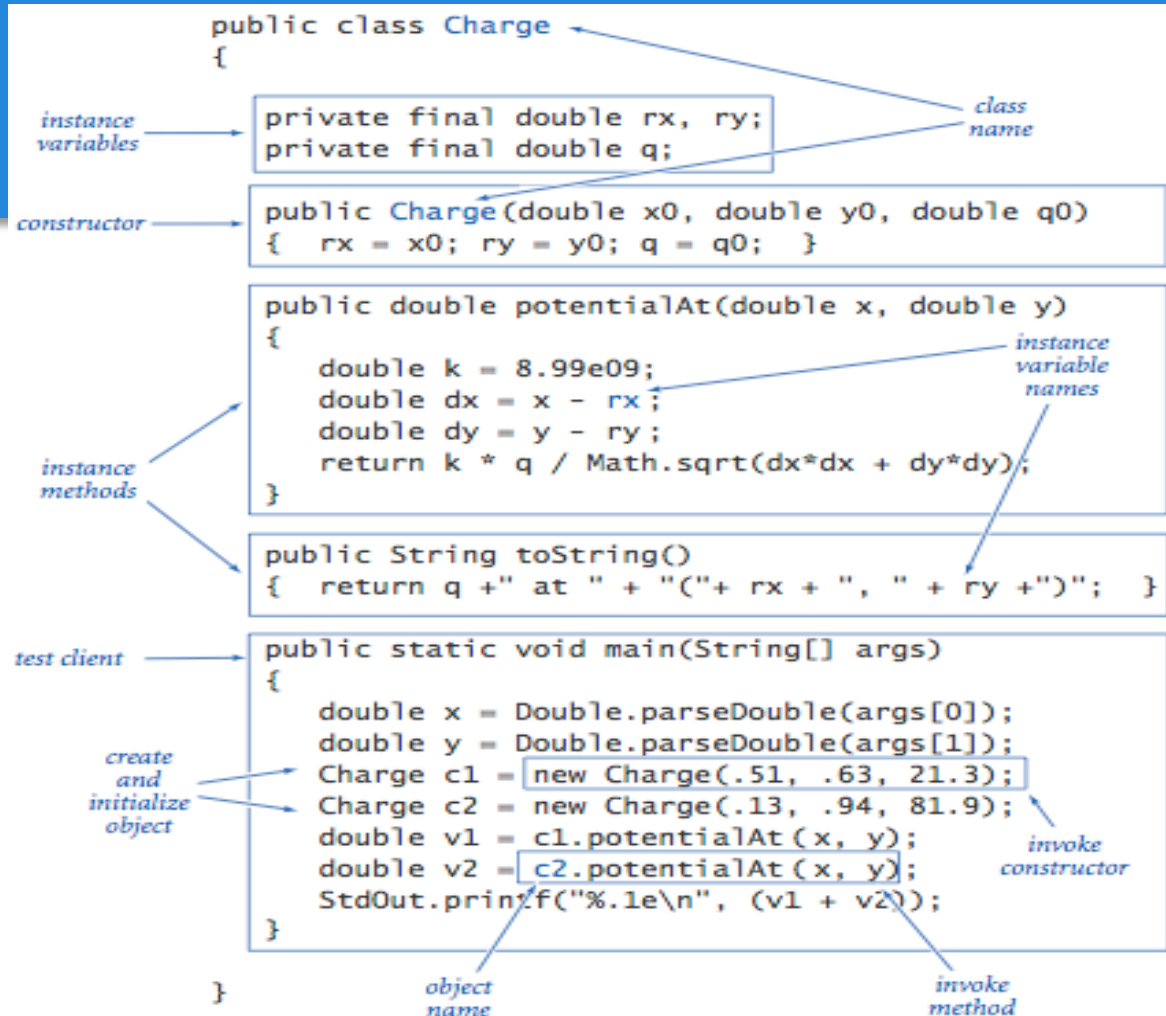
An object represents an entity in the real world that can be distinctly identified. An object is an instance of a class.

Circle Object 1

Data Fields:
radius is 1

Circle Object 2

Data Fields:
radius is 25



Classes and objects

Constructor with superclass initializer

Syntax

```
public ClassName(parameterType parameterName, . . .)
{
    super(arguments);
    . . .
}
```

The superclass constructor is called first.

```
public ChoiceQuestion(String questionText)
{
    super(questionText);
    choices = new ArrayList<String>;
}
```

The constructor body can contain additional statements.

If you omit the superclass constructor call, the superclass constructor with no arguments is invoked.

super keyword:

Invokes the superclass constructor or a superclass method

Classes and objects

this keyword

1. To assign a parameter variable to an instance variable of the current object
2. To call a constructor from another constructor in the same class
3. To pass a reference of the current object to another object

```
1. public class Curtain {  
2.     Builder builder = new Builder();  
3.     builder.setWallType(this);  
4. }  
5. public class Builder {  
6.     public void setWallType(Curtain c) {...}  
7. }
```

```
1. public class Curtain extends PrivacyWall {  
2.     String color;  
3.     public void setColor(String color) {  
4.         this.color = color;  
5.     }  
6. }
```

```
1. public class Curtain extends PrivacyWall {  
2.     public Curtain(int length, int width) {}  
3.     public Curtain() {  
4.         this(10, 9);  
5.     }  
6. }
```

Contents

- Big Java
- Classes and objects
- Object-oriented programming (OOP)
- Interfaces
- Packages
- Google Java style

OOP

- **Encapsulation** is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- **Inheritance** is the process by which one object acquires the properties of another object.
- **Polymorphism** is a feature that allows one interface to be used for a general class of actions.

OOP – encapsulation

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, <i>through inheritance</i>	No	No
From any non-subclass class outside the package	Yes	No	No	No

Access modifiers determine whether other classes can use a particular field or invoke a particular method.

OOP – inheritance

abstract classes

```
1. public abstract class Alarm {  
2.     public void reset() {...}  
3.     public abstract void renderAlarm();  
4. }
```

abstract methods

```
1. public class DisplayAlarm extends Alarm {  
2.     public void renderAlarm() {  
3.         System.out.println("Active alarm.");  
4.     }  
5. }
```

- An abstract class is typically used as a base class and cannot be instantiated.
- An abstract class can contain abstract and nonabstract methods, and it can be a subclass of an abstract or a nonabstract class.
- An abstract method contains only the method declaration, which must be defined by any nonabstract class that inherits it

OOP – inheritance

Overriding

```
1. public class CircleObject extends
   SimpleGeometricObject {
2. @Override
3. public String toString() {
4.     return super.toString() +
       "\nradius is " + radius;
5. }
6. }
```


Overloading

```
1. public static double max(double num1, double
   num2) {
2.     if (num1 > num2)
3.         return num1;
4.     else
5.         return num2;
6. }
7. public static double max(double num1, double
   num2, double num3) {
8.     return max(max(num1, num2), num3);
9. }
```

- Overridden methods are in different classes related by inheritance.
- Overloaded methods can be either in the same class or different classes related by inheritance.
- Overridden methods have the same signature and return type.
- Overloaded methods have the same name but a different parameter list.

OOP – inheritance

```
1. public class CircleObject extends SimpleGeometricObject
   {
2. @Override
3.     public String toString() {
4.         return super.toString() + "\nradius is " + radius;
5.     }
6. }
```



- An *annotation* is an instance of an annotation type and associates metadata with an application element.
- It is expressed in source code by prefixing the type name with the @ symbol.
- @Override annotations are useful for expressing that a subclass method overrides a method in the superclass and doesn't overload that method instead.

OOP – inheritance

Prevent overriding

```
1. class A {
2.     final void meth() {
3.         System.out.println("This is a final
         method.");
4.     }
5. }
6. class B extends A {
7.     void meth() { // ERROR! Can't override.
8.         System.out.println("Illegal!");
9.     }
10. }
```

Prevent inheritance

```
1. final class A {
2.     // ...
3. }
4. // The following class is illegal.
5. class B extends A { // ERROR! Can't
    subclass A
6. // ...
7. }
```

final keyword:

A value that cannot be changed after it has been initialized, a method that cannot be overridden, or a class that cannot be extended

OOP – polymorphism

```
1. public class PolymorphismDemo {
2.     public static void main(String[] args) {
3.         // Display circle and rectangle properties
4.         displayObject(new CircleFromSimpleGeometricObject(1, "red", false));
5.         displayObject(new RectangleFromSimpleGeometricObject(1, 1, "black",
6.             true));
7.     }
8.     public static void displayObject(SimpleGeometricObject object) {
9.         System.out.println("Created on " + object.getDateCreated()
10.            + ". Color is " + object.getColor());
11.     }
12. }
```

```
Created on Wed Sep 17 11:41:21 BST 2014. Color is red
Created on Wed Sep 17 11:41:21 BST 2014. Color is black
```

- The inheritance relationship enables a subclass to inherit features from its superclass with additional new features.
- A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa.

Contents

- Big Java
- Classes and objects
- Object-oriented programming (OOP)
- **Interfaces**
- Packages
- Google Java style

Interfaces

Syntax

```
public interface InterfaceName
{
    method headers
}
```

The methods of an interface are automatically public.

```
public interface Measurable
{
    double getMeasure();
}
```

No implementation is provided.

Syntax

```
public class ClassName implements InterfaceName, InterfaceName, . . .
{
    instance variables
    methods
}
```

```
public class BankAccount implements Measurable
{
    . . .
    public double getMeasure()
    {
        return balance;
    }
    . . .
}
```

List all interface types that this class implements.

BankAccount
instance variables

This method provides the implementation for the method declared in the interface.

Other
BankAccount methods

Interfaces

	Abstract class	Interface
Variables	No restrictions	All variables must be public static final .
Constructors	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No constructors. An interface cannot be instantiated using the new operator.
Methods	No restrictions	All methods must be public abstract instance methods

Interfaces

Inner classes

```
1. public class MeasurerTester {  
2.     public static void main(String[] args)  
3.     {  
4.         class AreaMeasurer implements Measurer  
5.         {  
6.             . . .  
7.             Measurer areaMeas = new AreaMeasurer();  
8.             double averageArea = Data.average(rects,  
9.             areaMeas);  
10.         }  
11. }
```

- Inner classes are commonly used for utility classes that should not be visible elsewhere in a program.
- An event listener belongs to a class that is provided by the application programmer. Its methods describe the actions to be taken when an event occurs.

Event listeners

```
1. import java.awt.event.ActionEvent;  
2. import java.awt.event.ActionListener;  
3. public class ClickListener implements  
4.     ActionListener {  
5.     public void actionPerformed(ActionEvent  
6.         event) {  
7.         System.out.println("I was clicked.");  
8.     }  
9. }
```

Contents

- Big Java
- Classes and objects
- Object-oriented programming (OOP)
- Interfaces
- Packages
- Google Java style

Packages

- Java provides a method of organizing the code in your programming projects into logical modules, or collections of code, called Packages.
- Packages are declared at the top of each Java code module that utilizes their classes and methods contained within those packages.
- Packages are always declared using Java's package keyword.

```
1 package com.example.mycontactlist;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.SQLException;
7 import android.database.sqlite.SQLiteDatabase;
8
9 public class ContactDataSource {
10     private SQLiteDatabase database;
11     private ContactDBHelper dbHelper;
12
13     public ContactDataSource(Context context) {
14         dbHelper = new ContactDBHelper(context);
15     }
16
17     public void open() throws SQLException {
18         database = dbHelper.getWritableDatabase();
19     }
20
21     public void close() {
22         dbHelper.close();
23     }
24 }
25
```

Google Java style

- **Classes and interfaces**
 - The first letter should be capitalized, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "camelCase").
- **Methods**
 - The first letter should be lowercase, and then normal camelCase rules should be used. In addition, the names should typically be verb-noun pairs.
- **Variables**
 - Like methods, the camelCase format should be used, starting with a lowercase letter.
- **Constants**
 - Java constants are created by marking variables static and final. They should be named using uppercase letters with underscore characters as separators.