JAVA PROGRAMMING 1

Summer 2017 - Christian Hur

Unit 1 Lecture - Introduction to Java

## What is Java?

Java is a powerful, versatile, general-purpose programming language for developing software running on mobile devices, desktop computers, and servers.

* Developed by team led by James Gosling at Sun Microsystems in 1991 (bought by Oracle in 2010)
* Originally Java was called “Oak” in 1991 - use in embedded chips in consumer electronic appliances
* 1995 - renamed to Java because Oak was already taken.
* Java Applets - standalone applications that run on Web server (web page)
* Java Library - the API that contains predefined classes and interfaces for developing Java programs.
* Java Development Toolkit (JDK) - the software for developing and running Java programs
* Java Language Specification defines the Java language’s syntax and semantics (<http://docs.oracle.com/javase/specs/>)
* Java is modeled after C++.
* “**Write once, run anywhere**” (**WORA**) is the slogan developed by Sun Microsystems to describe the ability of one Java program version to work correctly on multiple platforms.

## Editions and Versions of Java

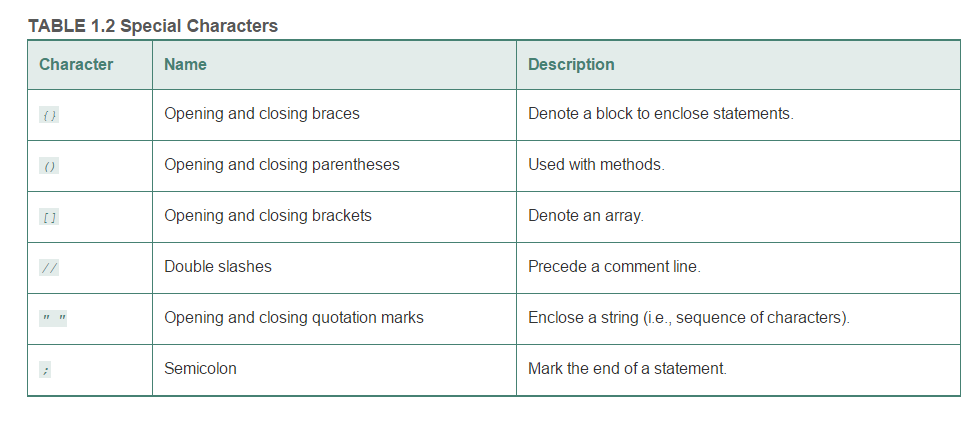
Java comes in three editions:

1. **Java Standard Edition** (Java SE) - develop client-side applications. Standalone or applets
2. **Java Enterprise Edition** (Java EE) - develop server-side applications such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
3. **Java Micro Edition** (Java ME) - develop applications for mobile devices such as cell phones.

Latest version: 8 or JDK 1.8.x

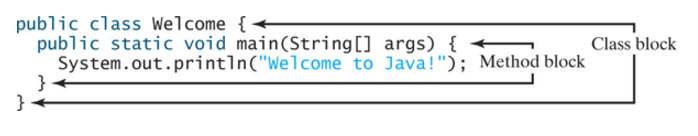
## Syntax

* A Java program is executed from the “main” method in the class.
* Java language is case sensitive (**area** , **Area** , and **AREA** are all different identifiers)
* Strongly-typed language
* Java source code is stored in a .java file
* Java source .java is compiled by the compiler (javac.exe) to .class file (bytecode)
* Bytecode is executed by Java Virtual Machine (JVM) on any system.

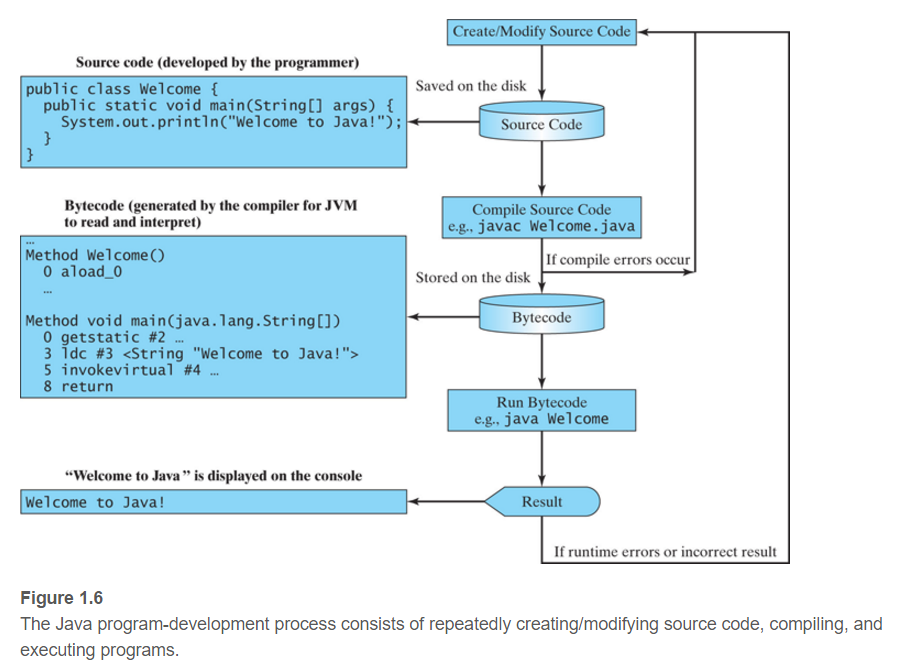


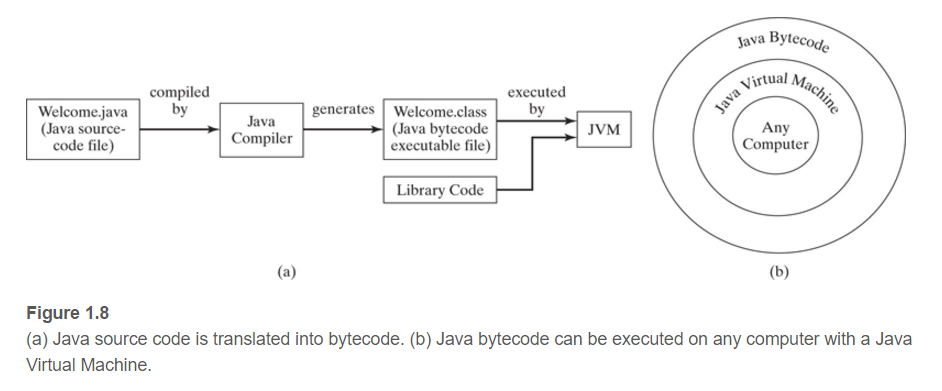
## Simple Java Program

A computer program is a set of instructions that tell a computer (machine) to perform a task. A program is composed of program statements. In order for a program to run (execute), it must be translated by a special program called a “compiler” or an “interpreter”. Consider the following simple Java program.



Below is a pictorial of the processes of how the simple program is compiled, interpretted, and executed.





# Compiler: javac.exe

The command to compile a Java program is “javac”. The interpreter to execute a compiled Java program is “java”.

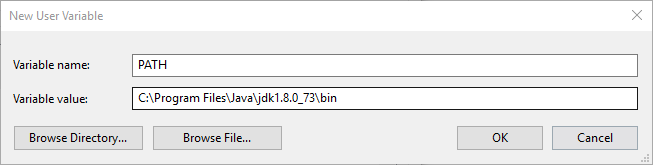
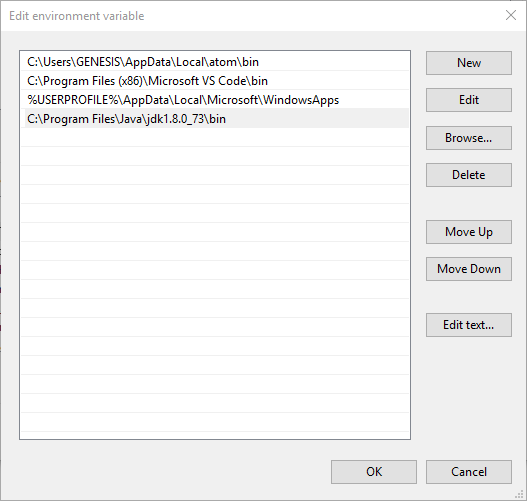
## Setup Environment Variables:

In order to compile and execute a Java program using the Command line, follow one of the following steps on your local machine to setup the environment.

### A. Command Prompt

1. Locate the path to javac.exe in JDK/bin.
2. C:> set path=”C:\Program Files\Java\jdk1.8.0\_73\bin”
3. Navigate to the location of your Java program (i.e. HelloWorld.java)
4. javac HelloWorld.java //Compiles it to HelloWorld.class
5. java HelloWorld //Executes - notice it doesn’t include the “class” extension

### B. GUI (Advanced System Settings)

1. Locate the path to javac.exe in JDK/bin.
2. Right click on “Computer” => Properties
3. Select “Advanced system settings”
4. Under the “Advanced” tab, click “Environment Variables…”
5. Click “New” if no PATH variable exists  
   
6. Click “Edit” the PATH variable if exists.
7. Click OK to exit out of all windows.
8. Restart CMD
9. Javac HelloWorld.java //Compile
10. Java HelloWorld //Execute

## Identifiers

* An identifier is a sequence of characters that consists of letters, digits, underscores ( \_ ), and dollar signs ( $ ).
* An identifier must start with a letter, an underscore ( \_ ), or a dollar sign ( $ ). It cannot start with a digit.
* An identifier cannot be a reserved word
* An identifier cannot be true , false , or null .
* An identifier can be of any length.

## Named Constants

* A named constant is an identifier that represents a permanent value.
* A constant must be declared and initialized in the same statement.
* The keyword “final” is used for declaring a constant

final double PI = 3.14;

## Naming Conventions

* Use lowercase for variables and methods. If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word—for example, the variables radius and area and the method print .
* Capitalize the first letter of each word in a class name—for example, the class names ComputeArea and System .
* Capitalize every letter in a constant, and use underscores between words—for example, the constants PI and MAX\_VALUE .

## I/O - Reading Input from Console

Java uses System.in to read input from console (keyboard), and System.out to write to console (monitor).

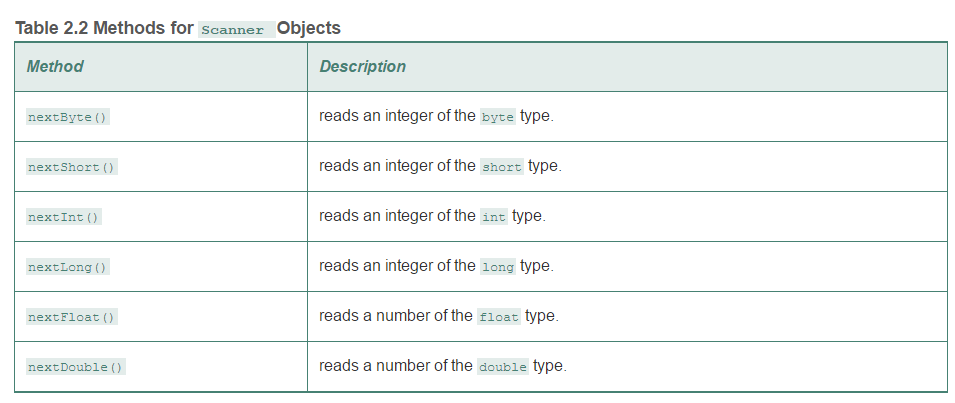
Output: System.out.print(“Hello world!”);

Input: Use the Scanner class to create an object to read input from the System.in.

Scanner input = new Scanner(System.in);

double radius = input.nextDouble();

int age = input.nextInt();



# 

# Code Indentation Styles

The curly braces, {}, are used in pairs to define a class and a method body. Note that in a properly formatted Java program, the curly brace that opens a method should be placed either at the end of the method header line or on the next line. Whitespace in the code is ignored by the compiler. The **K & R style** and **Allman style** of brace placement styles are good to use, and you should pick one that you understand.

|  |  |
| --- | --- |
| **Brace placement** | **Styles** |
| **while** (x == y) {  something();  somethingelse(); } | [K&R](https://en.wikipedia.org/wiki/Indent_style#K.26R) and variants:  [1TBS](https://en.wikipedia.org/wiki/Indent_style#1TBS), [Stroustrup](https://en.wikipedia.org/wiki/Indent_style#Variant:_Stroustrup), [Linux kernel](https://en.wikipedia.org/wiki/Indent_style#Variant:_Linux_kernel), [BSD KNF](https://en.wikipedia.org/wiki/Indent_style#Variant:_BSD_KNF) |
| **while** (x == y) {  something();  somethingelse(); } | [Allman](https://en.wikipedia.org/wiki/Indent_style#Allman_style) |
| **while** (x == y)  {  something();  somethingelse();  } | [GNU](https://en.wikipedia.org/wiki/Indent_style#GNU_style) |
| **while** (x == y)  {  something();  somethingelse();  } | [Whitesmiths](https://en.wikipedia.org/wiki/Indent_style#Whitesmiths_style) |
| **while** (x == y) { something();  somethingelse(); } | [Horstmann](https://en.wikipedia.org/wiki/Indent_style#Horstmann_style) |
| **while** (x == y) { something();  somethingelse(); } | [Pico](https://en.wikipedia.org/wiki/Indent_style#Pico_style) |
| **while** (x == y) {  something();  somethingelse();  } | [Ratliff](https://en.wikipedia.org/wiki/Indent_style#Ratliff_style) |
| **while** (x == y) {  something();  somethingelse(); } | [Lisp](https://en.wikipedia.org/wiki/Indent_style#Lisp_style) |

# Objects and Classes

*A class defines the properties and behaviors for objects.*

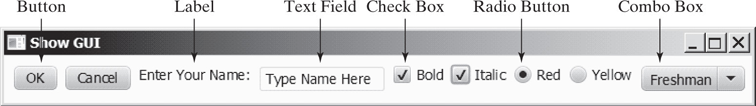
*Object-oriented programming (OOP) involves programming using objects. An object represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.*

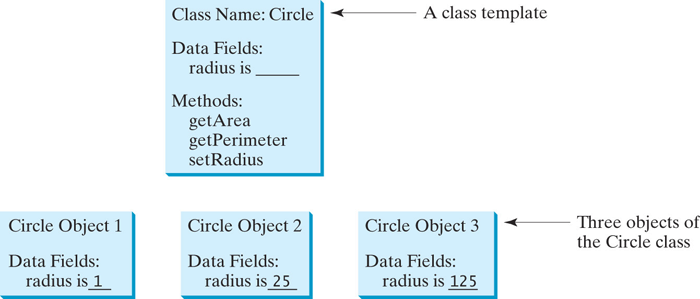
*An object has a unique* ***identity****,* ***state****, and* ***behavior****.*

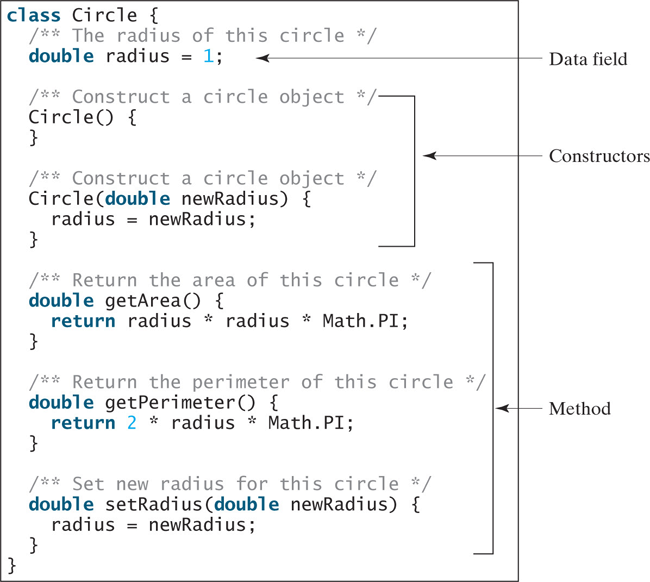
*The state of an object (also known as its properties or attributes) is represented by data fields with their current values.*

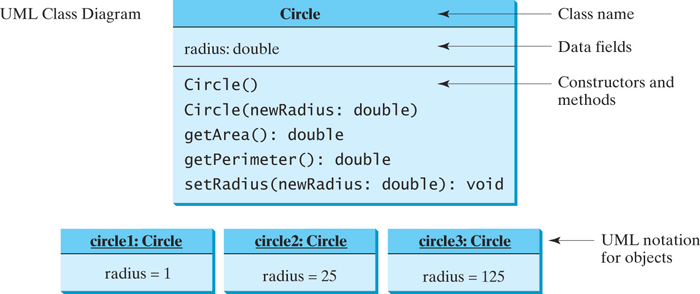
*The* ***behavior*** *of an object (also known as its actions) is defined by methods.*

*A* ***class*** *is a* ***template****, blueprint, or contract that defines what an object’s data fields and methods will be. An* ***object*** *is an* ***instance*** *of a class.*

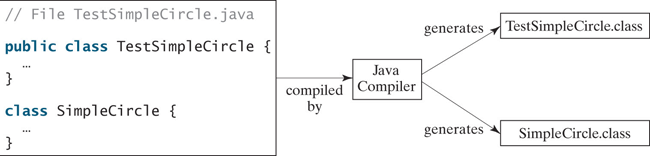


****





Each class in the source code file is compiled into a .class file.



## TV.java

1 public class TV {  
 2 int channel = 1; // Default channel is 1  
 3 int volumeLevel = 1; // Default volume level is 1  
 4 boolean on = false; // TV is off  
 5   
 6 public TV() {  
 7 }  
 8   
 9 public void turnOn() {  
10 on = true;  
11 }  
12   
13 public void turnOff() {  
14 on = false;  
15 }  
16   
17 public void setChannel(int newChannel) {  
18 if (on && newChannel >= 1 && newChannel <= 120)  
19 channel = newChannel;  
20 }  
21   
22 public void setVolume(int newVolumeLevel) {  
23 if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)  
24 volumeLevel = newVolumeLevel;  
25 }  
26   
27 public void channelUp() {  
28 if (on && channel < 120)  
29 channel++;  
30 }  
31   
32 public void channelDown() {  
33 if (on && channel > 1)  
34 channel—–;  
35 }  
36   
37 public void volumeUp() {  
38 if (on && volumeLevel < 7)  
39 volumeLevel++;  
40 }  
41   
42 public void volumeDown() {  
43 if (on && volumeLevel > 1)  
44 volumeLevel—–;  
45 }  
46 }

## TestTV.java (main program)

1 public class TestTV {   
 2 public static void main(String[] args) {  
 3 TV tv1 = new TV();  
 4 tv1.turnOn();  
 5 tv1.setChannel(30);   
 6 tv1.setVolume(3);   
 7   
 8 TV tv2 = new TV();  
 9 tv2.turnOn();  
10 tv2.channelUp();  
11 tv2.channelUp();  
12 tv2.volumeUp();  
13   
14 System.out.println("tv1's channel is " + tv1.channel   
15 + " and volume level is " + tv1.volumeLevel);  
16 System.out.println("tv2's channel is " + tv2.channel   
17 + " and volume level is " + tv2.volumeLevel);  
18 }  
19 }

## Data Field Encapsulation

*Making data fields private protects data and makes the class easy to maintain.*

*A private data field cannot be accessed by an object from outside the class that defines the private field. However, a client often needs to retrieve and modify a data field. To make a private data field accessible, provide a* ***getter method to return its value****. To enable a private data field to be updated, provide a* ***setter method to set a new value****.*

* *A getter method is also referred to as an accessor*
* *A setter method is referred to as a mutator.*

*Getter: returnType*

*public returnType getPropertyName()*

/\*\* Return radius \*/  
 public double getRadius() {  
 return radius;  
 }

*Setter: void*

*public void setPropertyName(dataType propertyValue)*

/\*\* Set a new radius \*/  
 public void setRadius(double newRadius) {  
 radius = (newRadius >= 0) ? newRadius : 0;  
 }

## Immutable Objects and Classes

*You can define immutable classes to create immutable objects. The contents of immutable objects cannot be changed.*

*If a class is immutable, then all its data fields must be private and it cannot contain public setter methods for any data fields. A class with all private data fields and no mutators is not necessarily immutable.*

*For a class to be immutable, it must meet the following requirements:*

* *All data fields must be private.*
* *There can’t be any mutator methods for data fields.*
* *No accessor methods can return a reference to a data field that is mutable.*

## The this Reference

*The keyword this refers to the object itself. It can also be used inside a constructor to invoke another constructor of the same class.*

## String, StringBuilder, and StringBuffer Classes

*The StringBuilder and StringBuffer classes are similar to the String class except that the String class is* ***immutable****.*

*In general, the StringBuilder and StringBuffer classes can be used wherever a string is used. StringBuilder and StringBuffer are more flexible than String . You can add, insert, or append new contents into StringBuilder and StringBuffer objects, whereas the value of a String object is fixed once the string is created.*

String str = “Larry Bird”;  
str = str + “ is a Professional Basketball player.”;

#### // str gets recreated and rewritten

StringBuilder stringBuilder = new StringBuilder();  
stringBuilder.append("Welcome");

stringBuilder.append(' ');  
stringBuilder.append("to");  
stringBuilder.append(' ');  
stringBuilder.append("Java");

#### //stringBuilder can append new characters without recreating the value

## Inheritance and Polymorphism

*Object-oriented programming allows you to define new classes from existing classes. This is called inheritance.*

*Inheritance enables you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses).*

*The keyword super refers to the superclass and can be used to invoke the superclass’s methods and constructors.*

*Example:*

public [[arrow]]CircleFromSimpleGeometricObject(  
 double radius, String color, boolean filled) {  
 super(color, filled);  
 this.radius = radius;  
}

1 public class Faculty extends Employee {  
 2 public static void main(String[] args) {  
 3 new Faculty();  
 4 }  
 5   
 6 public Faculty() {  
 7 System.out.println("(4) Performs Faculty's tasks");  
 8 }  
 9 }  
10   
11 class Employee extends Person {  
12 public Employee() {  
13 this("(2) Invoke Employee's overloaded constructor");  
14 System.out.println("(3) Performs Employee's tasks ");  
15 }  
16   
17 public Employee(String s) {  
18 System.out.println(s);  
19 }  
20 }  
21   
22 class Person {  
23 public Person() {  
24 System.out.println("(1) Performs Person's tasks");  
25 }  
26 }

## Calling Superclass Methods

The keyword *super* can also be used to reference a method other than the constructor in the superclass. The syntax is:

public void [[arrow]]printCircle() {  
 System.out.println("The circle is created " +   
 super.getDateCreated() + " and the radius is " + radius);  
}

*Overriding Methods*

*To override a method, the method must be defined in the subclass using the same signature and the same return type as in its superclass.*

*A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass. This is referred to as method overriding.*

1 public class CircleFromSimpleGeometricObject   
2 extends SimpleGeometricObject {  
3 // Other methods are omitted  
4   
5 // Override the toString method defined in the superclass

6 public String toString() {  
7 return super.toString() + "\nradius is " + radius;  
8 }  
9 }

## 

## 

## Overriding vs. Overloading

* *Overloading means to define multiple methods with the same name but different signatures.*
* *Overriding means to provide a new implementation for a method in the subclass.*

*Overriding:*

public class Test {  
 public static void main(String[] args) {  
 A a = new A();  
 a.p(10);   
 a.p(10.0);   
 }  
}  
   
class B {  
 public void p(double i) {  
 System.out.println(i \* 2);   
 }  
}  
   
class A extends B {  
 // This method overrides the method in B  
 public void p(double i) {  
 System.out.println(i);   
 }  
}

*Overloading:*

public class Test {  
 public static void main(String[] args) {  
 A a = new A();  
 a.p(10);   
 a.p(10.0);

A.total = 12;

a.p(“Hi”);  
 }  
}  
   
public B {

Double total;  
 public void p(double i) {

total += i;  
 System.out.println(i \* 2);   
 }

public void p(String s) {  
 System.out.println(s);   
 }

}  
   
class A extends B {  
 // This method overloads the method in B  
 public void p(int i) {  
 System.out.println(i);   
 }

@Override

public void p(String s) {  
 System.out.println(super.p(s));   
 }  
  
}

## Polymorphism

*The three pillars of object-oriented programming are encapsulation, inheritance, and polymorphism.*

*Polymorphism means that a variable of a supertype can refer to a subtype object.*

*For example, every circle is a geometric object, but not every geometric object is a circle. Therefore, you can always pass an instance of a subclass to a parameter of its superclass type.*

1 public class PolymorphismDemo {   
 2 /\*\* Main method \*/   
 3 public static void main(String[] args) {   
 4 // Display circle and rectangle properties   
 5 displayObject(new CircleFromSimpleGeometricObject   
 6 (1, "red", false));   
 7 displayObject(new RectangleFromSimpleGeometricObject   
 8 (1, 1, "black", true));   
 9 }   
10   
11 /\*\* Display geometric object properties \*/   
12 public static void displayObject(SimpleGeometricObject object) {   
13 System.out.println("Created on " + object.getDateCreated() +   
14 ". Color is " + object.getColor());   
15 }   
16 }