

互联网应用开发技术

Web Application Development

第7课

JAVA&JAVA EE

Episode Seven

Java and Java EE

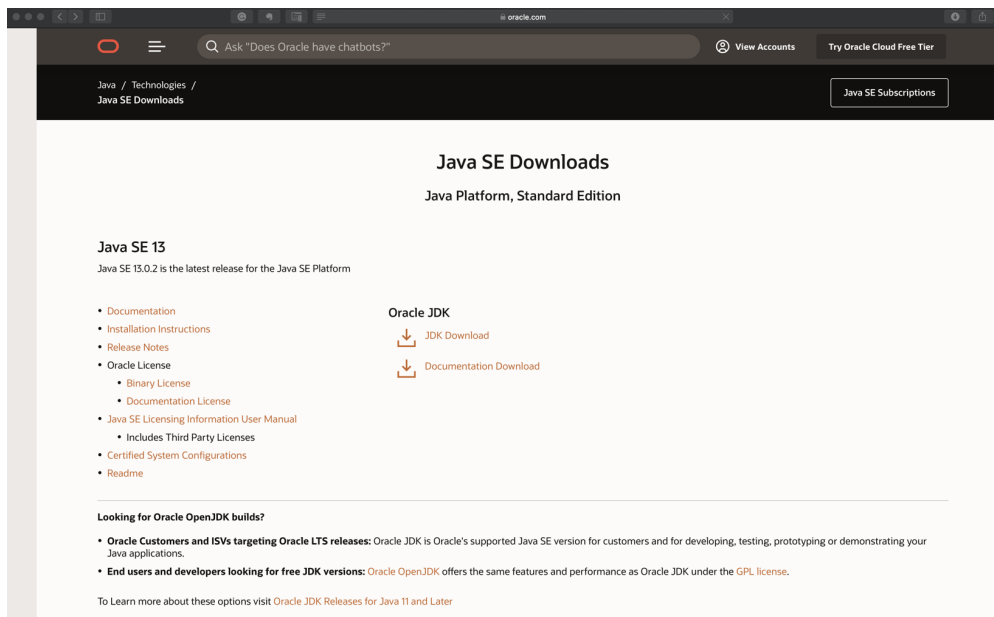
陈昊鹏

chen-hp@sjtu.edu.cn

Web Application
Development

- The Java Tutorials
 - Trail: Learning the Java Language
- Object Oriented Design
 - SOLID Principles
 - Package Principles
 - Layers & MVC
- Java EE
 - Java EE architecture

- Java
 - It was initiated in 1991 by James Gosling at Sun Microsystems as one of his many set-top box projects.
 - is an *object-oriented programming language*.



A Sample Java Program

```
import java.util.Scanner;

public class FirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello out there.");
        System.out.println("I will add two numbers for you.");
        System.out.println("Enter two whole numbers on a line:");

        int n1, n2;

        Scanner keyboard = new Scanner(System.in);

        n1 = keyboard.nextInt();
        n2 = keyboard.nextInt();

        System.out.println("The sum of those two numbers is");
        System.out.println(n1 + n2);
    }
}
```

Gets the Scanner class from the package (library) java.util

Name of the class—your choice. "This program should be in a file named FirstProgram.java"

Sends output to screen

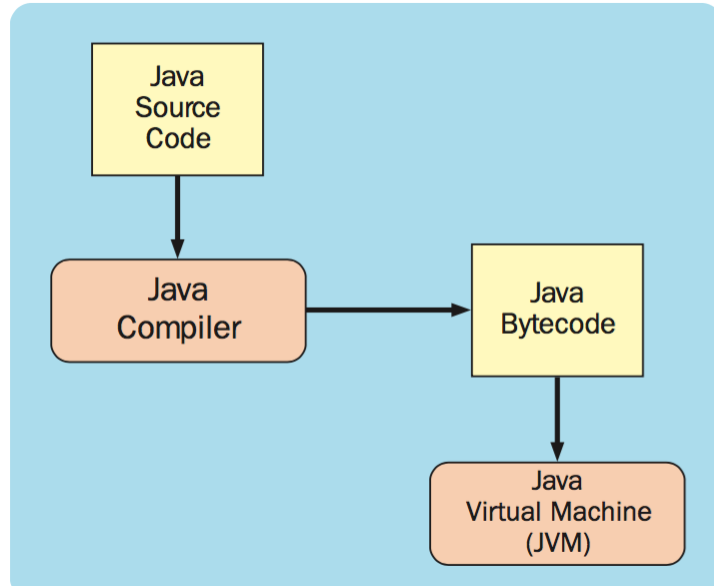
Says that n1 and n2 are variables that hold integers (whole numbers)

Readies the program for keyboard input

Reads one whole number from the keyboard

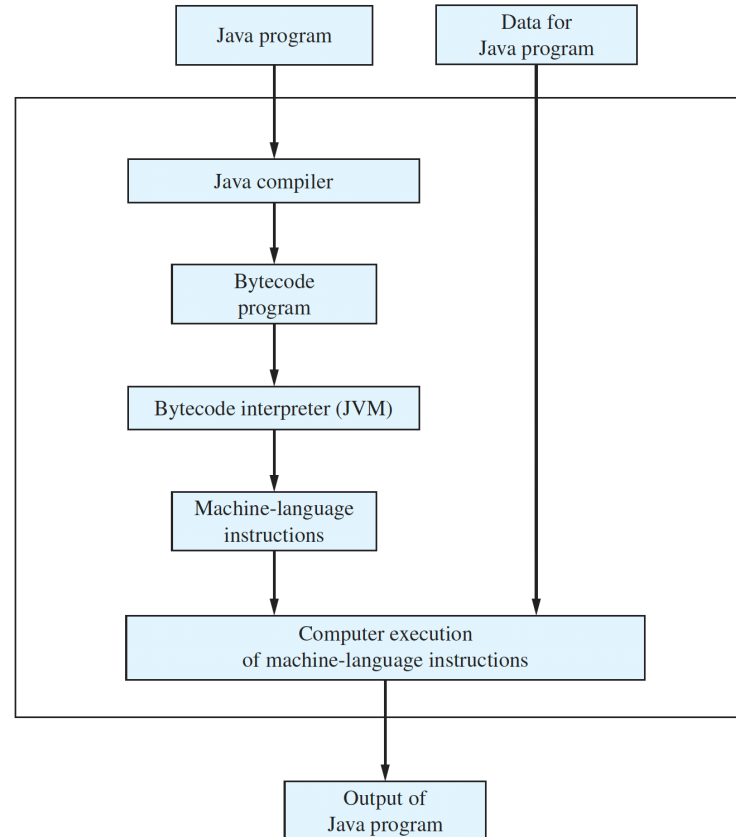
Sample Screen Output

```
Hello out there.
I will add two numbers for you.
Enter two whole numbers on a line:
12 30
The sum of those two numbers is
42
```

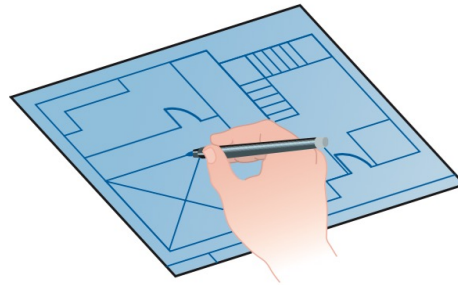


- Bytecode
 - The Java compiler translates your Java program into a language called bytecode.
 - This bytecode is not the machine language for any particular computer, but it is similar to the machine language of most common computers.
 - Bytecode is easily translated into the machine language of a given computer.
 - Each type of computer will have its own translator—called an interpreter—that translates from bytecode instructions to machine-language instructions for that computer. (JVM)

Compiling and Running a Java Program

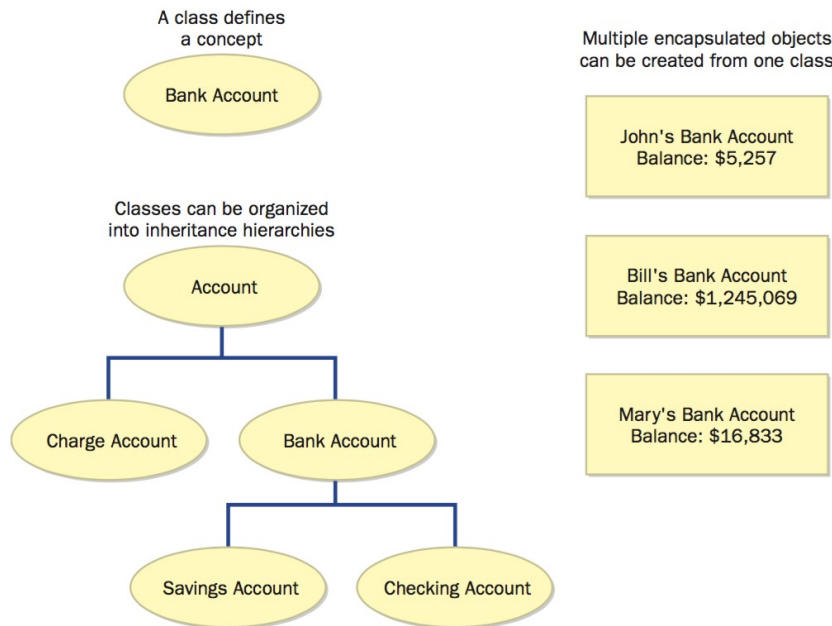


- Object-oriented programming ultimately requires a solid understanding of the following terms:
 - Object
 - Attribute
 - Method
 - Class
 - Encapsulation
 - Inheritance
 - Polymorphism

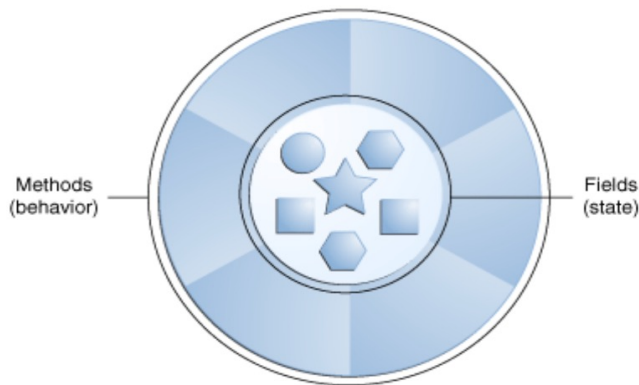


- Object-oriented programming ultimately requires a solid understanding of the following terms:

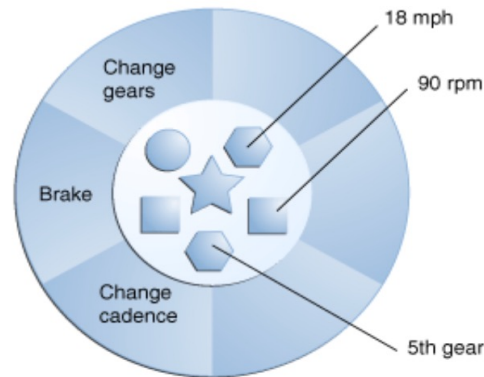
- Object
- Attribute
- Method
- Class
- Encapsulation
- Inheritance
- Polymorphism



- Real-world objects share two characteristics:
 - They all have *state* and *behavior*.



A software object.



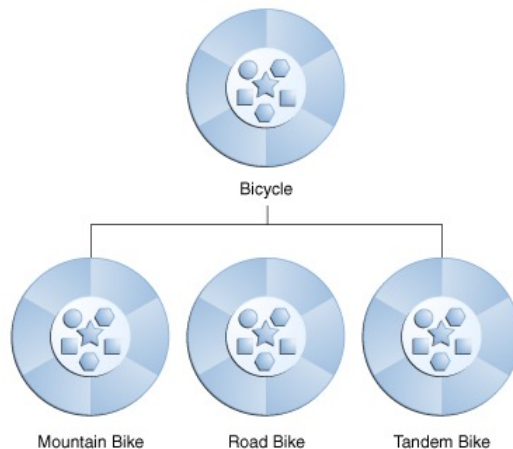
A bicycle modeled as a software object.

- A *class* is the blueprint from which individual objects are created.

```
class Bicycle {  
  
    int cadence = 0;  
    int speed = 0;  
    int gear = 1;  
  
    void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
  
    void printStates() {  
        System.out.println("cadence:" +  
            cadence + " speed:" +  
            speed + " gear:" + gear);  
    }  
}
```

```
class BicycleDemo {  
    public static void main(String[] args) {  
  
        // Create two different  
        // Bicycle objects  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle();  
  
        // Invoke methods on  
        // those objects  
        bike1.changeCadence(50);  
        bike1.speedUp(10);  
        bike1.changeGear(2);  
        bike1.printStates();  
  
        bike2.changeCadence(50);  
        bike2.speedUp(10);  
        bike2.changeGear(2);  
        bike2.changeCadence(40);  
        bike2.speedUp(10);  
        bike2.changeGear(3);  
        bike2.printStates();  
    }  
}
```

- Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.



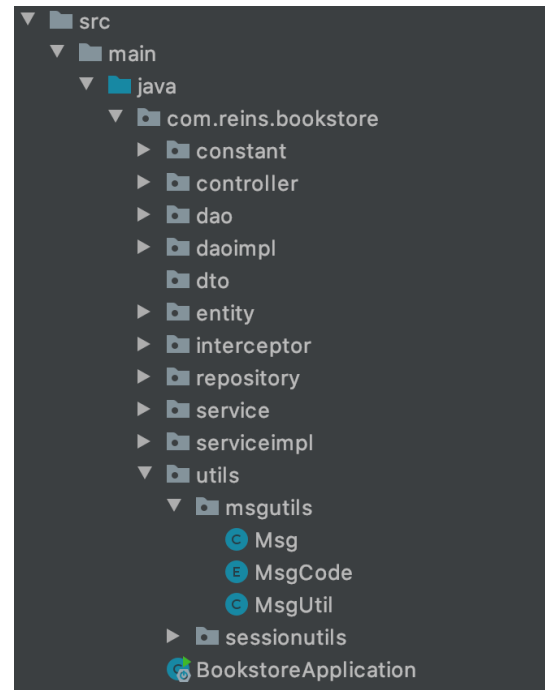
```
class MountainBike extends Bicycle {  
  
    // new fields and methods defining  
    // a mountain bike would go here  
  
}
```

- An *interface* is a group of related methods with empty bodies

```
interface Bicycle {  
  
    // wheel revolutions per minute  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
}
```

```
class ACMEBicycle implements Bicycle {  
  
    int cadence = 0;  
    int speed = 0;  
    int gear = 1;  
  
    // The compiler will now require that methods  
    // changeCadence, changeGear, speedUp, and applyBrakes  
    // all be implemented. Compilation will fail if those  
    // methods are missing from this class.  
  
    void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
  
    void printStates() {  
        System.out.println("cadence:" +  
            cadence + " speed:" +  
            speed + " gear:" + gear);  
    }  
}
```

- A package is a namespace that organizes a set of related classes and interfaces.
 - Conceptually you can think of packages as being similar to different folders on your computer.



- The Java programming language is **statically-typed**, which means that all variables must first be declared before they can be used.

- **byte**: The byte data type is an 8-bit signed two's complement integer.
- **short**: The short data type is a 16-bit signed two's complement integer.
- **int**: By default, the int data type is a 32-bit signed two's complement integer.
- **long**: The long data type is a 64-bit two's complement integer.
- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point.
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point.
- **boolean**: The boolean data type has only two possible values: true and false.
- **char**: The char data type is a single 16-bit Unicode character.

- Literals:

```
boolean result = true;  
char capitalC = 'C';  
byte b = 100;  
short s = 10000;  
int i = 100000;
```

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Operator Precedence

Operators	Precedence
postfix	<i>expr</i> ++ <i>expr</i> --
unary	++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=


```
int cadence = 0;  
anArray[0] = 100;  
System.out.println("Element 1 at index 0: " + anArray[0]);
```

```
int result = 1 + 2; // result is now 3  
if (value1 == value2)  
    System.out.println("value1 == value2");
```

```
class BlockDemo {  
    public static void main(String[] args) {  
        boolean condition = true;  
        if (condition) { // begin block 1  
            System.out.println("Condition is true.");  
        } // end block one  
        else { // begin block 2  
            System.out.println("Condition is false.");  
        } // end block 2  
    }  
}
```

Control Flow Statements

```
class IfElseDemo {
    public static void main(String[] args) {

        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        System.out.println("Grade = " + grade);
    }
}
```

```
class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1: case 3: case 5:
            case 7: case 8: case 10:
            case 12:
                numDays = 31;
                break;
            case 4: case 6:
            case 9: case 11:
                numDays = 30;
                break;
            case 2:
                if (((year % 4 == 0) &&
                    !(year % 100 == 0))
                    || (year % 400 == 0))
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Invalid month.");
                break;
        }

        System.out.println("Number of Days = "
            + numDays);
    }
}
```

```
class WhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        while (count < 11) {  
            System.out.println("Count is: " + count);  
            count++;  
        }  
    }  
}
```

```
class DoWhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        do {  
            System.out.println("Count is: " + count);  
            count++;  
        } while (count < 11);  
    }  
}
```

```
class ForDemo {  
    public static void main(String[] args){  
        for(int i=1; i<11; i++){  
            System.out.println("Count is: " + i);  
        }  
    }  
}
```

```
class EnhancedForDemo {  
    public static void main(String[] args){  
        int[] numbers =  
            {1,2,3,4,5,6,7,8,9,10};  
        for (int item : numbers) {  
            System.out.println("Count is: " + item);  
        }  
    }  
}
```

```
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 has  
    // one field  
    public int seatHeight;  
  
    // the MountainBike subclass has  
    // one constructor  
    public MountainBike(int startHeight, int startCadence,  
                        int startSpeed, int startGear) {  
        super(startCadence, startSpeed, startGear);  
        seatHeight = startHeight;  
    }  
  
    // the MountainBike subclass has  
    // one method  
    public void setHeight(int newValue) {  
        seatHeight = newValue;  
    }  
}
```

```
public class CreateObjectDemo {  
  
    public static void main(String[] args) {  
  
        // Declare and create a point object and two rectangle objects.  
        Point originOne = new Point(23, 94);  
        Rectangle rectOne = new Rectangle(originOne, 100, 200);  
        Rectangle rectTwo = new Rectangle(50, 100);  
  
        // display rectOne's width, height, and area  
        System.out.println("Width of rectOne: " + rectOne.width);  
        System.out.println("Height of rectOne: " + rectOne.height);  
        System.out.println("Area of rectOne: " + rectOne.getArea());  
  
        // set rectTwo's position  
        rectTwo.origin = originOne;  
  
        // display rectTwo's position  
        System.out.println("X Position of rectTwo: " + rectTwo.origin.x);  
        System.out.println("Y Position of rectTwo: " + rectTwo.origin.y);  
  
        // move rectTwo and display its new position  
        rectTwo.move(40, 72);  
        System.out.println("X Position of rectTwo: " + rectTwo.origin.x);  
        System.out.println("Y Position of rectTwo: " + rectTwo.origin.y);  
    }  
}
```

- There are two levels of access control:
 - At the top level—public, or *package-private* (no explicit modifier).
 - At the member level—public, private, protected, or *package-private* (no explicit modifier).

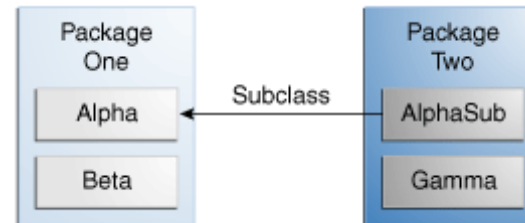
Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

- The following table shows where the members of the Alpha class are visible for each of the access modifiers that can be applied to them.

Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N



- Class Variables

```
public class Bicycle {  
  
    private int cadence;  
    private int gear;  
    private int speed;  
  
    // add an instance variable for the object ID  
    private int id;  
  
    // add a class variable for the  
    // number of Bicycle objects instantiated  
    private static int numberOfBicycles = 0;  
    ...  
}
```

Bicycle.numberOfBicycles

myBike.numberOfBicycles

- Class Methods

```
public static int getNumberOfBicycles() {  
    return numberOfBicycles;  
}
```

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods **cannot** access instance variables or instance methods directly—they must use an object reference. Also, class methods cannot use the `this` keyword as there is no instance for this to refer to.

- Constants

```
static final double PI = 3.141592653589793;
```

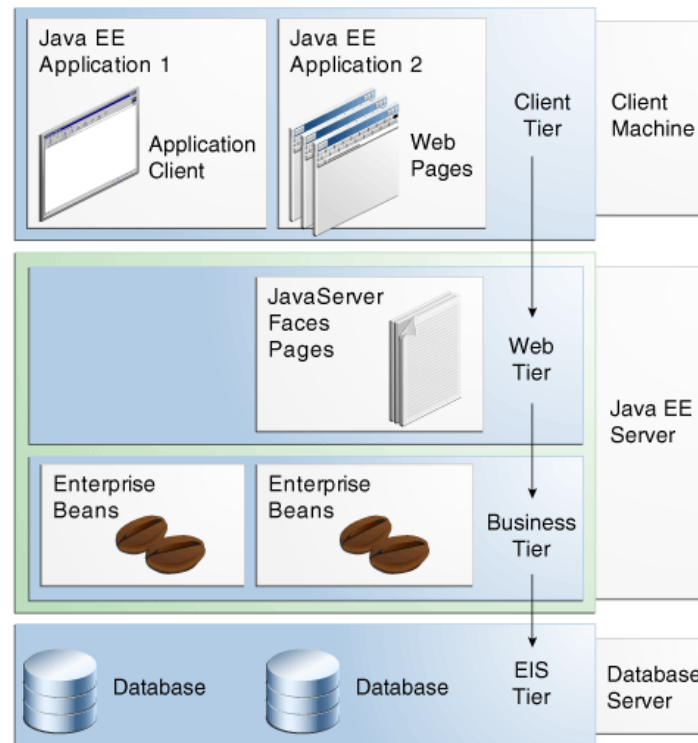
- What's the output of following program:

```
Change.java
1  public class Change{
2      public static void main(String args[]){
3          System.out.print(2.00 - 1.10);
4      }
5  }
```

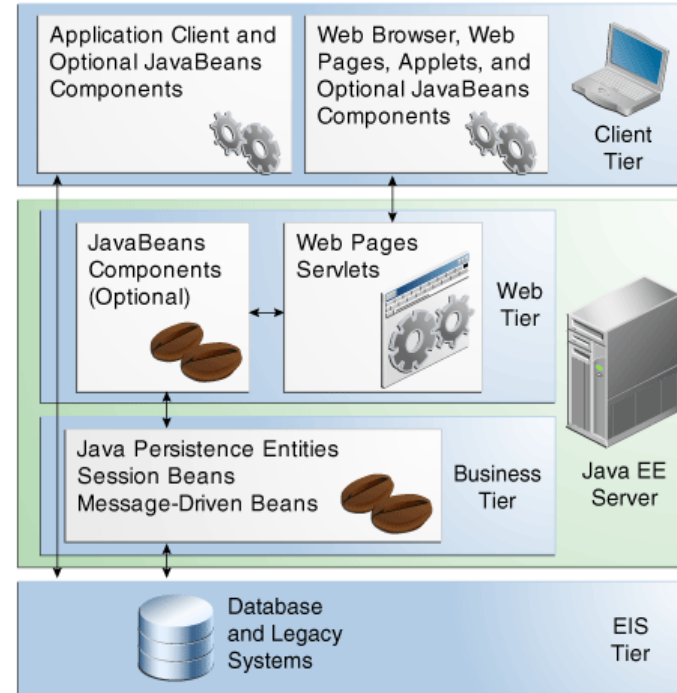
- Can this program be compiled?

```
BrowserTest.java  ✕
1  public class BrowserTest {
2      public static void main(String[] args) {
3          System.out.print("iexplore:");
4          http://www.google.com;
5          System.out.println(":maximize");
6      }
7  }
```

- The Java EE application parts shown in left figure are presented in Java EE Components.
 - Client-tier components
 - run on the client machine.
 - Web-tier components
 - run on the Java EE server.
 - Business-tier components
 - run on the Java EE server.
 - Enterprise information system (EIS)-tier software
 - runs on the EIS server.
- Ali, eBay, LinkedIn



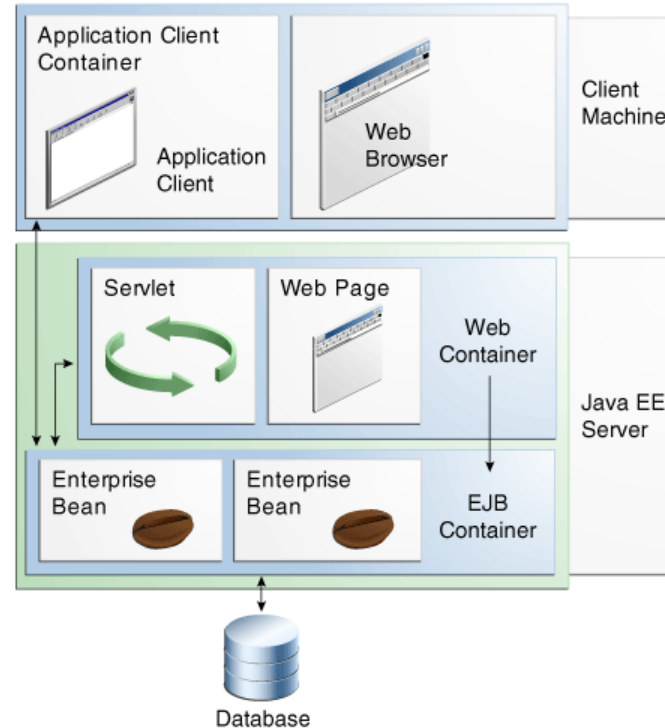
- The Java EE application parts shown in left figure are presented in Java EE Components.
 - Client-tier components
 - run on the client machine.
 - Web-tier components
 - run on the Java EE server.
 - Business-tier components
 - run on the Java EE server.
 - Enterprise information system (EIS)-tier software
 - runs on the EIS server.



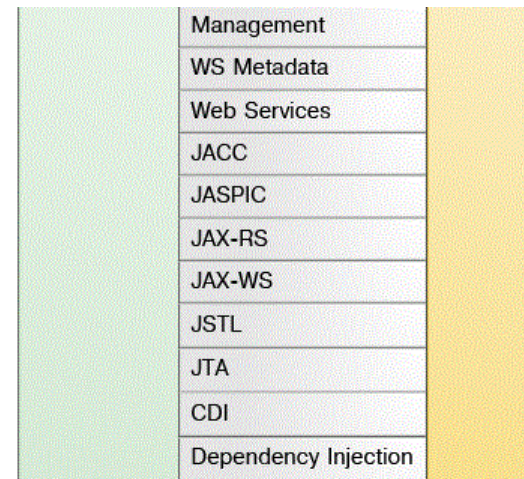
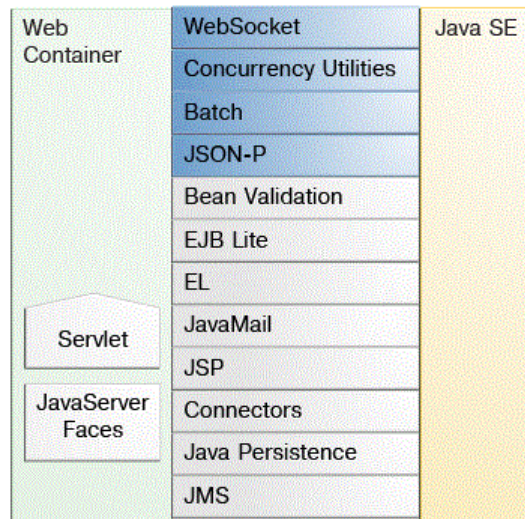
- Containers


- are the interface between a component and the low-level platform-specific functionality that supports the component.
- Before it can be executed, a web, enterprise bean, or application client component must be assembled into a Java EE module and deployed into its container.
- Container settings customize the underlying support provided by the Java EE server, including such services as
 - Security
 - Transaction management
 - Java Naming and Directory Interface (JNDI) API lookups,
 - and remote connectivity.

- Container types



- Java EE APIs in the Web Container



 New in Java EE 7

- JAVE SOFTWARE SOLUTIONS: FOUNDATIONS OF PROGRAM DESIGN (*Eighth Edition*)
 - JOHN LEWIS(Virginia Tech), WILLIAM LOFTUS (Accenture)
- Java: An Introduction to Problem Solving & Programing(Eighth Edition)
 - Walter Savitch(University of California, San Diego)
- The Java™ Tutorials -Trail: Learning the Java Language
 - <https://docs.oracle.com/javase/tutorial/java/index.html>
- SOLID
 - <https://en.wikipedia.org/wiki/SOLID>
- Package principles
 - https://en.wikipedia.org/wiki/Package_principles
- Principles of Package and Component Design
 - <https://blog.avenuecode.com/principles-of-package-and-component-design>
- The Java EE 8 Tutorial – Distributed Multitiered Applications
 - <https://javaee.github.io/tutorial/overview004.html#BNAAY>



- *Web*开发技术
- *Web Application Development*

Thank You!