Java 2

2009.10.13
Variables
Operators
Expressions, Statements and Blocks
Control Flow Statements

LANGUAGE BASICS
Variables

• **Instance Variables**
  – Non-Static Fields

• **Class Variables**
  – Static Fields

• **Local Variables**

• **Parameters**
  – Pass by value
  – Pass by reference
Primitive Data Types

- byte – 8 bit int ( -128 ~ 127)
- Short – 16 bit
- Int – 32 bit
- Long – 64 bit
- Float – 32 bit
- Double – 64 bit
- boolean
- Char – 16 bit
- String - [java.lang.String class](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html)
Array

- An array is a container object that holds a fixed number of values of a single type.
- Example:
  ```java
  int[] anArray; // declares an array of integers
  anArray = new int[10]; // allocates memory for 10 integers
  anArray[0] = 100; // initialize first element
  anArray[1] = 200; // initialize second element
  ```
OPERATORS
Postfix  \( expr++ \) \( expr-- \)
unary \( ++expr \) \( --expr \) \(+expr \) \(-expr\) \(\sim\) \(!\)
multiplicative \(*\) \(/\) \(%\)
additive \(+\) \(-\)
shift \(<<\) \(>>\) \(>>>\)
relational \(<\) \(\leq\) \(\geq\) \(==\) \(!=\)
equality \(==\) \(!=\)
bitwise AND \&
bitwise exclusive OR \(^\)
bitwise inclusive OR \(|\)
logical AND \(&&\)
logical OR \(||\)
ternary \(?:\)
assignment \(=\) \(+=\) \(-=\) \(*=\) \(\%=\) \(&=\) \(^=\) \(|=\) \(<<=\) \(>>=\) \(>>>=\)
The instanceof operator compares an object to a specified type. You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.

```java
Parent obj1 = new Parent();
Parent obj2 = new Child();
System.out.println("obj1 instanceof Parent: " + (obj1 instanceof Parent));
System.out.println("obj1 instanceof Child: " + (obj1 instanceof Child));
```

Output:
obj1 instanceof Parent: true
obj1 instanceof Child: false
Bitwise, Bit shift

- "<<" shifts a bit pattern to the left,
- ">>" shifts a bit pattern to the right.
- ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.
- & operator performs a bitwise AND operation.
- ^ operator performs a bitwise exclusive OR operation.
- | operator performs a bitwise inclusive OR operation.
EXPRESSIONS, STATEMENTS, & BLOCKS
EXPR, STAT, BLOCK

• Expression and statements
  – \((x+y)/100\)
  – \(Z = (x+y)/100\)

• Blocks
  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
FLOW CONTROL STATEMENTS
Control-Flow Statements

• Decision-making statements
  – (if-then, if-then-else, switch),

• Looping statements
  – (for, while, do-while),

• Branching statements
  – (break, continue, return)
CLASSES AND OBJECTS
• class MyClass {
    // fields, constructor, and method declarations
}
• 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; }
}
• class MyClass extends MySuperClass implements YourInterface {
   //field, constructor, and method declarations
}

• 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;
   }
}
Method

• Method declaration example
  public double calculateAnswer(double wingSpan, int numberOfEngines,
       double length, double grossTons) {
       //do the calculation here
  }

• overloading methods
  – Java can distinguish between methods with different method signatures.
    public class DataArtist {
      ...
      public void draw(String s) { ... }
      public void draw(int i) { ... }
      ...
    }

Constructors

• constructors are invoked to create objects from the class blueprint

• A class can have multiple constructors
  
  public Bicycle(int startCadence, int startSpeed, int startGear) {
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
  }
  
  Bicycle myBike = new Bicycle(30, 0, 8);

  public Bicycle( ) {
    gear = 1;
    cadence = 10;
    speed = 0;
  }
  
  Bicycle yourBike = new Bicycle( );
Objects

• public class Point {
    public int x = 0;
    public int y = 0;
    //constructor
    public Point(int a, int b) {
        x = a;
        y = b;
    }
}

→ create objects:
    Point originOne = new Point(23, 94);
Object in object

• public class Rectangle {
  public int width = 0;
  public int height = 0;
  public Point origin;
  //constructor
  public Rectangle(Point p, int w, int h) {
    origin = p;
    width = w;
    height = h;
  }
}

→ Create objects:
  Rectangle rectOne = new Rectangle(originOne, 100, 200);
Use Objects

• Within its own class
  – System.out.println("Width and height are: "+width+", "+height);

• Outside the class
  – System.out.println("Width of rectOne: "+rectOne.width);
    System.out.println("Height of rectOne: "+rectOne.height);

• Calling an object’s methods
  – objectReference.methodName(argumentList);
    or
  – objectReference.methodName();
Garbage Collector

• Garbage Collection
  – The Java runtime environment deletes objects when it determines that they are no longer being used. This process is called *garbage collection*.
  – An object is eligible for garbage collection when there are no more references to that object.
  – The Java runtime environment has a garbage collector that periodically frees the memory used by objects that are no longer referenced.
Returning values from methods.

- return;
- return returnValue;
- return a * b;
- return class;

  - When a method uses a class C as its return type, the class of the type of the returned object must be either a subclass of, or the exact class of, the return type.
The this keyword.

```java
public class Point {
    public int x = 0;
    public int y = 0;
    //constructor public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```
Public, Private and Protected

- Recalled:
  - Member variables in a class—these are called *fields*.
  - Variables in a method or block of code—these are called *local variables*.
  - Variables in method declarations—these are called *parameters*.

- **public modifier**—the field is accessible from all classes.
  - public class bicycle{ public int cadence; .... }

- **private modifier**—the field is accessible only within its own class.
  - public class bicycle{ private int cadence; .... }

- Protected: package - private
### Access Levels

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class</th>
<th>Package</th>
<th>Subclass</th>
<th>World</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>protected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><em>no modifier</em></td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>private</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Note: subclass here is declared outside the package
shows where the members of the **Alpha** class are visible for each of the access modifiers that can be applied to them.
The keyword `static`

- Static Variables
  - public class bicycle{ private static int numberOfBike; .... }
  - Gear is a *static field* or a *class variable*
  - *All bicycle objects have the same* `numberOfBike` *variable*
  - `Bicycle.numberOfBike`

- **Static Methods**
  - *Static methods to access static variables*

- **Constants**
  - `static final` `double PI = 3.141592653589793;`
  - `final` means this field can’t be changed
Nested Classes

- Non-static inner class – have access to the members of outer class
- Static inner class – don’t have access to the members of outer class

```java
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```