codecademy

Object-Oriented Programming in Java

Access Modifiers

There are four types of access modifiers, each with different scopes:

- public
- protected
- no modifier
- private

The scope ranges from public classes, which are accessible from anywhere in the program, to private classes, which are only accessible in the class itself.

Modifier	Class	Package	Child Class	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

Objects

An object is a specific instance of a class that uses attributes and behaviors to represent a real-world entity.

```
class Whale {
  // Instance variables define the
state of an object:
  String whaleSpecies;
  int whaleWeight;

  // Constructor:
  public Whale(String name, int
weight) {
    // Assign values to instance
variables:
    whaleSpecies = name;
    whaleWeight = weight;
  }

  // Non-static method:
  public String whaleCall() {

System.out.println("EEEEOOOUUUUUAAAAAA
");
```

```
  }

  public static void main(String[]
args){
    // Create a Whale object:
    Whale whale1 = new
Whale("narwhal", 2100);
    whale1.whaleCall();
// Prints:
EEEEOOOUUUUUAAAAAA
  }
}
```

Accessors and Mutators

Accessors and mutators (or getters and setters) are public methods that are used to access and change `private` instance variables.

```
private String name;

// Accessor method:
public String getName() {
  return name;
}

// Mutator method:
public void setName(String newName) {
  this.name = newName;
}
```

Variable Scope

In Java, variables have three different levels of scope:

- class scope: instance variables, which are accessible throughout the class they are declared in
- method scope: local variables, which are only accessible in the method in which they are declared or passed into as an argument
- block scope: loop variables, which are only accessible in the loop or conditional in which they are declared

```
public class Car {
  public String color; // This
instance variable is accessible
throughout the class

  public void setColor(String
newColor) {
    // newColor is a local variable
that is only accessible in this method
  }

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

```
    public Car toyota = new Car();
toyota is also a local variable, and
is only accessible in this method

    for (char c : toyota.color) {
      // c is a loop variable that is
only accessible in this block
      char temp = c; // temp is also a
loop variable that is only accessible
in this block
    }
  }

}
```

Inheritance

Inheritance is the concept of allowing a class to inherit the methods and properties of another class.
Superclasses refer to the class that another class inherits from. Subclasses refer to the class that inherits qualities from another class.

```
// Parent class:
class Animal {
  String sound;
  Animal(String snd) {
    this.sound = snd;
  }
}

// Child class:
class Dog extends Animal {
  // super() method can act like the
parent constructor inside the child
class constructor.
  Dog() {
    super("woof");
  }
  // alternatively, we can override
the constructor completely by defining
a new constructor.
  Dog() {
    this.sound = "woof";
  }
}
```

Polymorphism

Polymorphism allows a subclass to share the information and behavior of its superclass while also incorporating its own functionality.

```
// Parent class:
class Monster {
  String monsterNoise;
  public Monster(String monsterNoise)
{
    this.monsterNoise = monsterNoise;
  }

  public void makeNoise() {
    System.out.println(monsterNoise);
  }
}

// Child class:
class Vampire extends Monster {
  public Vampire() {
    super("I want to steal your
blood!");
  }

  // Override the parent class method:
  @Override
  public void makeNoise() {
    System.out.println("Muahahaha " +
monsterNoise);
  }

  public static void main(String[]
args) {
    Monster dracula = new Vampire();
    dracula.makeNoise();
    // Prints: Muahahaha I want to
steal your blood!
  }
}
```