JAVA PROGRAMMING

Week 3: Classes, Objects and Methods

Lecturer:

- Hồ Tuấn Thanh, M.Sc.

Plan

Plan

1. Class fundamentals
2. Methods
3. Constructors
4. The new operator revisited
5. Garbage collection
6. The this keyword

Class: General Form

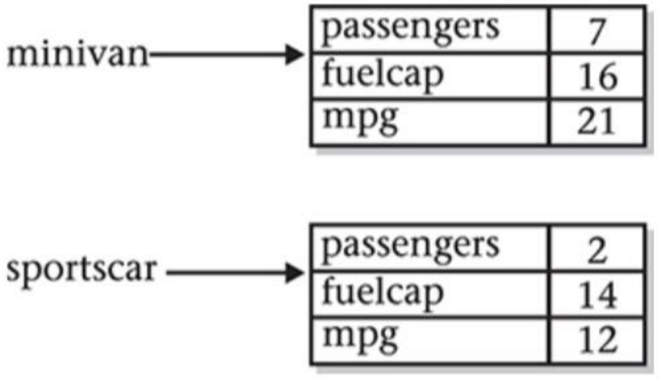
```
class Classname {
    // declare instance variables type var1;
    type var1;
    // ...
    type varN;
    // declare methods
    type method1(parameters) {
        // body of method
    }
    // ...
    type methodN(parameters) {
    // body of method
    }
}
```

Example: Defining a Class

```java
class Vehicle {
    int passengers; // number of passengers
    int fuelcap; // fuel capacity in gallons
    int mpg; // fuel consumption in miles per gallon
}
```

```java
public class VehicleDemo {
        public static void main(String[] args) {
                Vehicle minivan = new Vehicle();
                int range;
                // assign values to fields in minivan
                minivan.passengers = 7;
                minivan.fuelcap = 16; minivan.mpg = 21;
                // compute the range assuming a full tank of gas
                range = minivan.fuelcap * minivan.mpg;
                System.out.println("Minivan can carry " +
                        minivan.passengers +
                                        " with a range of " + range);
        }
}
```

```
...
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
minivan.passengers = 7;
minivan.fuelcap = 16; minivan.mpg = 21;
// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14; sportscar.mpg = 12;
// compute the range assuming a full tank of gas
range1 = minivan.fuelcap * minivan.mpg;
range2 = sportscar.fuelcap * sportscar.mpg;
System.out.println("Minivan can carry " + minivan.passengers
                              + " with a range of " + range1);
System.out.println("Sportscar can carry " + sportscar.passengers
                              + " with a range of " + range2);
```



How objects are created

```
Vehicle minivan = new Vehicle();
```

- This declaration performs two functions
  - declares a variable called minivan of the class type Vehicle.
  - creates an instance of the object and assigns to minivan a reference to that object. This is done by using the new operator.
- The statement can be rewritten:

```
Vehicle minivan; // declare reference to object
minivan = new Vehicle(); // allocate a Vehicle object
```

Reference variables and assignment

```java
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
car1.mpg = 26;
System.out.println("Car 1: " + car1.mpg);
System.out.println("Car 2: " + car2.mpg);
```

Result:

```
Car 1: 26
Car 2: 26
```

```java
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
Vehicle car3 = new Vehicle();
car2 = car3 ;
// now car2 and car3 refer to the same object
```

Plan

1. Class fundamentals
2. Methods
3. Constructors
4. The new operator revisited
5. Garbage collection
6. The this keyword

Method

```
ret-type name(parameter-list) {
        // body of method
}
```

- ret-type specifies the type of data returned by the method.
  - This can be any valid type, including class types that you create.
  - If the method does not return a value, its return type must be void.
- The name of the method is specified by name.
- The parameter-list is a sequence of type and identifier pairs separated by commas.

Adding a Method to the Vehicle Class

```java
class Vehicle1 {
	int passengers; // number of passengers
	int fuelcap; // fuel capacity in gallons
	int mpg; // fuel consumption in miles per gallon
	// Display the range
	void range() {
		System.out.println("Range is " + fuelcap * mpg);
	}
}
```

```
public class AddMethod {
        public static void main(String[] args) {
                Vehicle1 minivan = new Vehicle1();
                Vehicle1 sportscar = new Vehicle1();
                ...
                System.out.print("Minivan can carry " +
                                                minivan.passengers + ". ");
                minivan.range();
                System.out.print("Sportscar can carry " +
                                                sportscar.passengers + ". ");
                sportscar.range();
        }
}
```

Returning from a method

- In general, there are two conditions that cause a method to return:
  - when the method’s closing curly brace is encountered.
  - when a return statement is executed.
- There are two forms of return
  - return ; cause the immediate termination of a void method
  - returning values.

```
void myMethod() {
    for(int i = 0; i < 10; i++) {
        if (i == 5) return; // stop at 5
        System.out.println();
    }
}
```

Returning a value

```
// Use a return value.
class Vehicle2 {
        int passengers; // number of passengers
        int fuelcap; // fuel capacity in gallons
        int mpg; // fuel consumption in miles per gallon
        // Display the range
        int range() {
                return fuelcap * mpg;
        }
}
```

```
public class RetMethod {
        public static void main(String[] args) {
                Vehicle2 minivan = new Vehicle2();
                Vehicle2 sportscar = new Vehicle2();
                int range1, range2;
                ...
                //get the ranges
                range1 = minivan.range();
                range2 = sportscar.range();
                System.out.println("Minivan can carry " +
                  minivan.passengers + " with a range of " + range1);
                System.out.println("Sportscar can carry " +
                  sportscar.passengers + " with a range of " +
                  range2);
                }
}
```

Using parameters

- It is possible to pass one or more values to a method when the method is called.
- A value passed to a method is called an **argument**.
- Inside the method, the variable that receives the argument is called a **parameter**.
- Parameters are declared inside the parentheses that follow the method’s name.

```
class ChkNum{
        // return true if x is even
        boolean isEven(int x) {
                if ((x%2) == 0) return true;
                else return false;
        }
}
public class ParamDemo {
        public static void main(String[] args) {
                ChkNum e = new ChkNum();
                if(e.isEven(10)) System.out.println("10 is even.");
                if(e.isEven(9)) System.out.println("9 is even.");
                if(e.isEven(8)) System.out.println("8 is even.");
        }
}
```

```
class Factor{
        boolean isFactor(int a, int b) {
                if((b%a) == 0) return true;
                return false;
        }
}
public class IsFact {
        public static void main(String[] args) {
                Factor x = new Factor();
                if(x.isFactor(2, 20)) System.out.println(
                                                "2 is factor");
                if(x.isFactor(3, 20)) System.out.println(
                                                "This won't be displayed");
        }
}
```

Example: Adding a Parameterized Method

```
class Vehicle3 {
    int passengers; // number of passengers
    int fuelcap; // fuel capacity in gallons
    int mpg; // fuel consumption in miles per gallon
    // Return the range
    int getRange() {
        return mpg * fuelcap;
    }
    // Compute fuel needed for a given distance.
    double getFuelNeeded(int miles) {
        return (double) miles / mpg;
    }
}
```

```
public class CompFuel {
        public static void main(String[] args) {
                Vehicle3 minivan = new Vehicle3();
                Vehicle3 sportscar = new Vehicle3();
                double gallons;
                int dist = 252;
                ...
                gallons = minivan.getFuelNeeded(dist);
                System.out.println("To go " + dist +
                                " miles minivan needs " + gallons +
                                " gallons of fuels.");
                gallons = sportscar.getFuelNeeded(dist);
                System.out.println("To go " + dist +
                                " miles sportscar needs " + gallons +
                                " gallons of fuels.");
        }
}
```

Exercise: Improve Help example

```
class Help{
	void helpOn(int what) {
		// Display the help information based on a user's choice
	}
	void showMenu() {
		// Show menu option
	}
	boolean isValid(int choice) {
		// Check for a valid response
		return true;
	}
}
```

```java
public class HelpClassDemo {
        public static void main(String args[])
                                                        throws java.io.IOException{
                // Create an instance of Help class
                /* Invoke all the methods in that instance in order to:
                 *  - Display a menu,
                 *  - Input the user's choice, check for a valid
                 * response, and display information about the item
                 * selected.
                * The program also loops until the letter q is pressed.
                */
        }
}
```

Plan

Constructor

- A constructor initializes an object when it is created.
- It has the same name as its class and is syntactically similar to a method.
  - Constructors have no explicit return type.
  - Use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor.
- Once you define your own constructor, the default constructor is no longer used.

Example

```java
class MyClass{
        int x;
        MyClass(){
                x = 10;
        }
}
public class ConsDemo {
        public static void main(String[] args) {
                MyClass t1 = new MyClass();
                MyClass t2 = new MyClass();
                System.out.println(t1.x + " " + t2.x);
        }
}
```

Parameterized constructors

- Most often you will need a constructor that accepts one or more parameters.
- Parameters are added to a constructor in the same way that they are added to a method: just declare them inside the parentheses after the constructor's name.
- Example:

```
// A parameterized constructor
MyClass(int i){
        x = i;
}
....
MyClass t3 = new MyClass(10);
```

Example: Adding a constructor to the vehicle class

```
//This is a constructor for Vehicle
Vehicle4(int p, int f, int m){
        passengers = p;
        fuelcap = f;
        mpg = m;
}
....
Vehicle4 minivan = new Vehicle4(7, 16, 21);
Vehicle4 sportscar = new Vehicle4(2, 14, 12);
```

The new operator revisited

```
classvar = new Classname(arglist);
```

- classvar is a variable of the class type being created.
- classname is the name of the class that is being instantiated.

Garbage collection [1]

- Objects are dynamically allocated from a pool of free memory by using the new operator.
- Memory is not infinite, and the free memory can be exhausted.
  - It is possible for new to fail because there is insufficient free memory to create the desired object.

  → For this reason, a key component of any dynamic allocation scheme is the recovery of free memory from unused objects, making that memory available for subsequent reallocation.

→ garbage collection

Garbage collection [2]

- Java's garbage collection system reclaims objects automatically – occurring transparently, behind the scenes, without any programmer intervention.
- It works like this:
  - When no references to an object exist → that object is assumed to be no longer needed → the memory occupied by the object is released.
  - This recycled memory can then be used for a subsequent allocation.

The `this` keyword

- When a method is called, it is automatically passed an implicit argument that is a reference to the invoking object →This reference is called `this`.
- However, `this` has some important uses.
- Java syntax permits the name of a parameter or a local variable to be the same as the name of an instance variable.
  - When this happens, the local name hides the instance variable.
  - You can gain access to the hidden instance variable by referring to it through `this`.

```
class Pwr2{
        double base;
        int e;
        double val;
        Pwr2(double base, int exp){
                this.base = base;
                this.e = exp;
                this.val = 1;
                if(exp == 0) return;
                for(; exp > 0; exp--) this.val = this.val * base;
        }
        double getPwr() {
                return this.val;
        }
}
```

QUESTION ?