Chapter 8: Classes and Objects : A Deeper Look

TASNIM SHARMIN ALIN
LECTURER
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING
LEADING UNIVERSITY,SYLHET

Class Scope

- Class scope
  - Class variables and methods
  - Members are accessible to all class methods
  - Members can be referenced by name
    - objectReferenceName.objectMemberName
  - Shadowed (hidden) class variables
    - `this`.variableName

Controlling Access to Members

- Member access modifiers
  - Control access to class's variables and methods
  - `public`
    - Variables and methods accessible to clients of the class
  - `private`
    - Variables and methods not accessible to clients of the class

This keyword

- Keyword **`this`** (***`this`*** *reference*)
  - Allows an object to refers to itself

Here is given the 6 usage of this keyword.

- this keyword can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this keyword can also be used to return the current class instance.

Where to use this() constructor call?

- The this() constructor call should be used to reuse the constructor in the constructor.
- It maintains the chain between the constructors i.e. it is used for constructor chaining.

this keyword can be passed as an argument in the method.

```
public class S {
 void m(S obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
    public static void main(String[] args) {
        S s1 = new S();
        s1.p();
    }
}
```

Output :

method is invoked

The this keyword can be passed as argument in the constructor call.


```
class B{
  A obj;
  B(A obj){
    this.obj=obj;   }
  void display(){
    System.out.println(obj.data);     / /using data member of A class
  }
}
public class A {
   int data=10;
   A(){
   B b=new B(this);
   b.display();
  }
    public static void main(String[] args) {
          A a=new A();
    }
}
```


The this keyword can be used to return current class instance.

Syntax: return_type method_name(){
return this; }

```
class A{
A getA(){
return this;
}
void msg(){
      System.out.println("Hello java");}
}
public class Test {
   public static void main(String[] args) {

      new A().getA().msg();
   }
}
```

Output:

Hello java

Passing object as Parameter

```
class Rectangle{
   int length,width;
   Rectangle(int length,int width){
      this.length=length;
      this.width = width;
   }
   void area (Rectangle r1){
      int volumn = r1.length * r1.width;
      System.out.println("volumn of rectangle is : "+ volumn);
   }
}
public class RectangleDemo {
   public static void main(String[] args) {
      Rectangle r1= new Rectangle(10,20);
      r1.area(r1);
   }
}
```

Output: volumn of rectangle is : 200

Using Set and Get Methods

- Accessor method (“get” method)
  - `public` method
  - Allow clients to read `private` data
- Mutator method (“set” method)
  - `public` method
  - Allow clients to modify `private` data

Static Class Members

- `static` keyword
  - `static` class variable
    - Class-wide information
      - All class objects share same data
- Access to a class's `public static` members
  - Qualify the member name with the class name and a dot (.)
    - e.g., Math.random()

Static import

- A static import declaration enables programmers to refer to imported static members without the class name and a dot(.).
- A static import declaration has two forms:

• one that imports a particular static member which is known as sngle static import.

import static *packageName.ClassName.staticMemberName;*

• one that imports all static members of a class which is known as static import o demand.

import static *packageName.ClassName.*;*

Static import

```
import static java.lang.Math.*;
public class StaticImport {

   public static void main(String[] args) {
      System.out.printf("sqrt(900.0) = %.1f\n",sqrt(900.0));
      System.out.printf("ceil(-9.8) = %.1f\n",ceil(-9.8));
      System.out.printf("log(E) = %.1f\n",log(E));
      System.out.printf("cos(0.0) = %.1f\n",cos(0.0));
   }
}
```

Output:

sqrt(900.0) = 30.0

ceil(-9.8) = -9.0

log(E) = 1.0

cos(0.0) = 1.0

Final Instance Variables

- `final` keyword

- The **final keyword** in java is used to restrict the user.
  - Indicates that variable is not modifiable
    - Any attempt to modify `final` variable results in error

```
private final int INCREMENT = 5;
```

    - Declares variable `INCREMENT` as a *constant*

- The final keyword can be used in many context. Final can be:

- variable
- method
- class

Example of *final* variable

```
public class Bike {
  final int speedlimit=40;
  void run(){
     speedlimit=400;

  }
   public static void main(String[] args) {
      Bike obj=new Bike();
      obj.run();
   }
```

Output:

Compile Time Error

Example of *final* Method

```
class Bike{
  final void run(){
     System.out.println("running"); }
}
public class Honda extends Bike{
     void run(){
        System.out.println("running safely with 100kmph");
     }
    public static void main(String[] args) {
       Honda honda= new Honda();
       honda.run();
    }
}
```

Output:

Compile Time Error

- **Is final method inherited?**

Ans : Yes, final method is inherited but you cannot override it.

```
class Bike{
  final void run(){
     System.out.println ( "running.....");
  }
}
public class Honda extends Bike{
    public static void main(String[] args) {
        new Honda().run();
    }
}
```

Output:

```
running.....
```


- **If you make any class as final, you cannot extend it.**

```
final class Bike{
}
public class Honda1 extends Bike1 {
    void run(){
        System.out.println("running safely with 100kmph");
    }
   public static void main(String[] args) {
     new Honda().run();
   }
}
```

Output:

Compile Time Error

Is final method inherited?

- Yes, final method is inherited but you cannot override it.

```
class Bike{
 final void run(){
System.out.println("running...");}
 }
 class Honda extends Bike{
public static void main(String args[]){
    new Honda().run();
}
  }
```

Output: running...

Can we initialize blank final variable?

- Yes, we can initialize clank final variable but only in constructor.

```
public class Bike2 {
final int speedlimit;          //blank final variable

  Bike2(){
  speedlimit=70;
  System.out.println(speedlimit);
  }
    public static void main(String[] args) {
          new Bike2();
    }
}
```

output: 70

static blank final variable

- A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
public class A {
 static final int data;
  static{
     data=50;
  }
   public static void main(String[] args) {
       System.out.println(A.data);
   }
}
```

output: 50

- **Can we declare a constructor final?**

Ans: No, because constructor is never inherited.

Exercises

Chapter 8

Fig 8.14, 8.15, 8.16, 8.8