Web Technology 8

Abstract Class & Nested Class

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Abstract Method

Abstract Method

- Inheritance allows a sub-class to override the methods of its super-class
- In fact, a super-class may altogether leave the implementation details of a method and declare such a method abstract

  abstract type name(parameter-list);

- Two kinds of methods:
  - **concrete** - may be overridden by sub-classes
  - **abstract** - must be overridden by sub-classes
- It is illegal to define abstract constructors or static methods
- *abstract double area();*

Abstract Class

Abstract Class

- A class that contains an abstract method must be itself declared abstract
- An abstract class has no instances - it is illegal to use the new operator
- It is legal to define variables of the abstract class type
- A sub-class of an abstract class:
    - implements all abstract methods of its super-class, or
    - is also declared as an abstract class
- *Abstract super-class, concrete sub-class*

Abstract Class...

```
abstract class A {
        abstract void callme();
        void callmeto () {
                System.out.println("This is a concrete method.");
        }       }
class B extends A {
        void callme() {
                System.out.println("B's implementation.");
        }       }
class AbstractDemo {
        public static void main(String args[]) {
                B b = new B();
                b.callme();
                b.callmeto ();
                }
        }
```

Abstract Class...

```
abstract class Figure {
        double dim1, dim2;
        Figure(double a, double b) {
                dim1 = a; dim2 = b;
                }
        abstract double area();
   }
class Rectangle extends Figure {
        Rectangle(double a, double b) {
                        super(a, b);
                        }
        double area() {
                System.out.println("Inside Area for Rectangle.");
                return dim1 * dim2;
                }
        }
```

Abstract Class...

```
class Triangle extends Figure {
        Triangle(double a, double b) {
                super(a, b);
                }
        double area() {
                System.out.println("Inside Area for Triangle.");
                return dim1 * dim2 / 2;
                }
        }
class AbstractAreas {
        public static void main(String args[]) {
                Rectangle r = new Rectangle(9, 5);
                Triangle t = new Triangle(10, 8);
                Figure figref;
                figref = r; System.out.println(figref.area() );
                figref = t; System.out.println(figref.area() );
                }
        }
```

final keyword

final keyword

- The final keyword has three uses:
  - declare a variable which value cannot change after initialization
  - declare a method which cannot be overridden in sub-classes
  - declare a class which cannot have any sub-classes
- **final variable**:
  - Declaring a variable as final prevents its contents from being modified
    final int FILE_NEW=1;

final keyword...

final keyword...

- **final method**:
  - A method declared final cannot be overridden in any sub-class
    class A {
    final void meth() {
    System.out.println(This is a final method."); }
    }
  - As a final method cannot be overridden, their invocations are resolved at compile-time. This is one way to improve performance of a method call
- **final class**:
  - A class declared final cannot be inherited - has no sub-classes
    final class A {...}
  - *Declaring a class final implicitly declares all its methods final*

Object Class

Object Class

- Object class is a super-class of all Java classes
- Object is the root of the Java inheritance hierarchy
- A variable of the Object type may refer to objects of any class
- As arrays are implemented as objects, it may also refer to any array
  - Object clone( )
  - boolean equals(Object object)
  - void finalize( )
  - Class getClass( )
  - void notify( )
  - void notifyAll( )
  - void wait( )
  - void wait(long milliseconds)
  - void wait(long milliseconds, int nanoseconds)
  - String toString( )
- *All methods except getClass, notify, notifyAll and wait can be overridden*

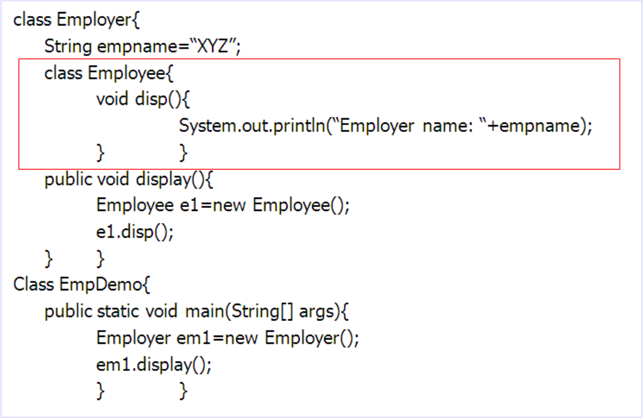
Nested Class

- A nested class is declared in the body of another class
- The nested class exists only as long as the outer class exists
- The scope of nested class is limited to the scope of the outer class

Inner Classes

Inner Classes

An inner class is a member of the outer class, just like any other members of the outer class. Thus, it has access to all the members of the outer class including the private members. However, the outer class does not have direct access to members of the inner class

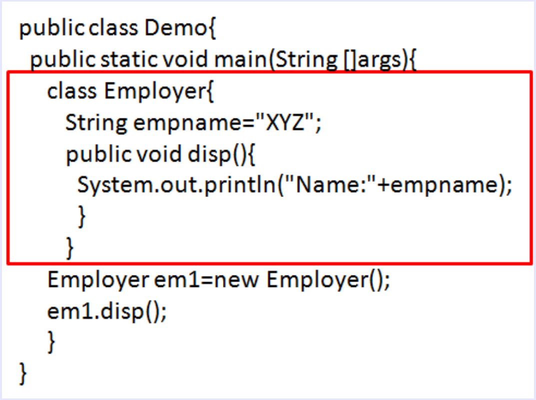


```
class Employer{
     String empname="XYZ";
     class Employee{
          void disp(){
               System.out.println("Employer  name: "+empname);
          }          }
     public void display(){
          Employee e1=new Employee();
          e1.disp();
     }     }
Class EmpDemo{
     public static void main(String[] args){
          Employer em1=new Employer();
          em1.display();
          }          }
```

Local Class

Local Class

A local class is declared in a block or a method. Thus, its scope is limited to the block or method. This class can refer to local variables or parameters which are declared final. It is not visible outside the block in which it is declared

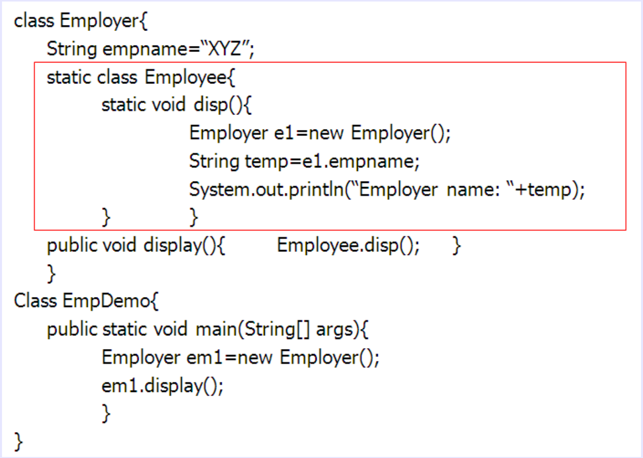


```
public class Demo{
 public static void main(String []args){
   class Employer{
     String empname="XYZ";
     public void disp(){
      System.out.println("Name:"+empname);
      }
     }
   Employer em1=new Employer();
   em1.disp();
   }
}
```

Static Nested Class

Static Nested Classe

A nested class is a member of outer class like any other static member. This class can access only the static instance variables or methods. The nested static class may be accessed inside the main method without the reference of outer class



```
class Employer{
      String empname=“XYZ”;
      static class Employee{
              static void disp(){
                          Employer e1=new Employer();
                          String temp=e1.empname;
                          System.out.println(“Employer  name: “+temp);
              }           }
      public void display(){        Employee.disp();     }
      }
Class EmpDemo{
      public static void main(String[] args){
              Employer em1=new Employer();
              em1.display();
              }
}
```