Module 6

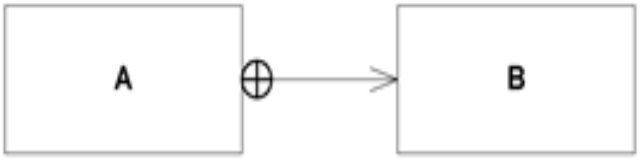
Inner and anonymous classes

Inner class

- A class defined within another class is referred to as nested or inner class.

- As a matter of fact, a class can be declared in any block, including blocks that are part of a method.

- What is so peculiar about nested classes is their visibility and accessibility.




```
public class A {
  private class B {
    ...
  }
}
```

Note! Classes nested in the method are called local.

Example of inner class

```
public class OuterOne {
     private int x;
     public class InnerOne {
          private int y;
          public void innerMethod() {
               System.out.println("enclosing x is " + x);
               System.out.println("y is " + y);
          }
     }
     public void outerMethod() {
          System.out.println("x is " + x);
     }
     public void makeInner() {
          InnerOne anInner = new InnerOne();
          anInner.innerMethod();
     }
     // other methods...
}
```

Inner class

- The full name of the inner class from the example is `OuterOne.InnerOne`.

- Upon compilation, a separate file named `OuterOne$InnerOne.class` is created.

- An inner class can only be instantiated through an outer class instance.

Note! All attributes of the outer class are available to the nested class, because there is a reference to outer class in the nested class (it will be covered later).

Inner class

- Inner classes can only be instantiated through an outer class instance.

```
public static void main(String[] args)
{
  OuterOne.InnerOne i = new OuterOne().new InnerOne();
  i.innerMethod();

  OuterOne outer = new OuterOne();
  OuterOne.InnerOne inner = outer.new InnerOne();
}
```

- Inner classes can be **public**, **private**, **protected** or with default access.

Static inner classes

- Inner class can be declared as **`static`**.

- A static nested class cannot use the **`this`** keyword to access outer object attributes.

- Yet, it can request static variables and static outer class methods.

```
public class MyOuter {
    public static class MyInner {
    }

    public static void main(String[] args) {
        MyInner aMyInner = new MyOuter.MyInner();
    }
}
```

Local and anonymous classes

- Anything declared within a method is not a class member.
- Local objects cannot have access modifiers and cannot be declared as `static`.
- Anonymous classes can be created.
- A local or anonymous class can only access outer objects if they are declared as `final`.

Local classes

```
public class MOuter {
    private int m = (int) (Math.random() * 100);
    public static void main(String args[]) {
        MOuter that = new MOuter();
        that.go((int) (Math.random() * 100), (int) (Math.random() *
100));
    }
    public void go(int x, final int y) {
        int a = x + y;
        final int b = x - y;
        class MInner {
            public void method() {
                System.out.println("m is " + m);
                // System.out.println("x is " + x); //Illegal!
                System.out.println("y is " + y);
                // System.out.println("a is " + a); //Illegal!
                System.out.println("b is " + b);
            }
        }
        MInner that = new MInner();
        that.method();
    }
}
```

Anonymous classes

- You can declare an inner class within the body of a method without naming it.

- Anonymous class can be declared as extension to another class or as an interface implementation.

Example:

- Closing of user window by clicking the “Close” button.

- The response is programmed as a separate class:

```
class CloseActionListener implements ActionListener
```

Anonymous classes

```
public class Xbis {
    public static void main( String[] args){
        JFrame frame = new JFrame("Closing example");
        JButton btnClose = new JButton("Close");

        CloseActionListener al = new CloseActionListener();
        btnClose.addActionListener( al ); // .addActionListener()

        frame.add(btnClose);
        frame.pack();
        frame.setVisible(true);
    } // main
} // class X

class CloseActionListener implements ActionListener{
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Closing...");
        System.exit(0);
    }
}
```

Anonymous classes

```
public static void main( String[] args){
   JFrame frame = new JFrame("Closing example");
   JButton btnClose = new JButton("Close");
   btnClose.addActionListener(
         new ActionListener(){

         @Override
         public void actionPerformed(ActionEvent e){
               System.out.println("Closing...");
               System.exit(0);
            }
         } // new ActionListener
         ); // .addActionListener()

   frame.add(btnClose);
   frame.pack();
   frame.setVisible(true);
} // main
```

Note! Inheritance and implementation cannot be used at a time.

Anonymous classes

- Anonymous classes are practical when you don’t want to use trivial names for classes.

- The class code contains several lines.

- When compiling an anonymous class, a separate class named `EnclosingClassName$n` is created, where `n` is an anonymous class order number in the outer class.

Anonymous classes

```
public void aMethod() {
   theButton.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e) {
         System.out.println("The action has
occurred");
        }
     }
   );
}
```

- A constructor cannot be defined for an anonymous class.

- The superclass constructor can be called.

Anonymous classes

```
public class X extends Frame {
 private int count;
 public X() {
   final Label label = new Label("Count = " + count);
   add(label, BorderLayout.SOUTH);
   Button button = new Button("Increase counter");

   button.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
         count++;
         label.setText("Count = " + count);
      }
   });

   add(button, BorderLayout.NORTH);
  }

}
```

Using of inner and anonymous classes

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

   List<Pet> pets = new ArrayList<>();
   RoboDog d = new RoboDog("Robik");
   pets.add(d);

   // create inner class
   class SpecialRoboDog extends RoboDog {
       public void beFriendly() {
           System.out.println(
                   "I'm very special for you!");
       }
   }

   pets.add(new SpecialRoboDog());

   // add anonymous class
   pets.add(new RoboDog() {
       public void beFriendly() {
           System.out.println(
            "I'm more friendly than everyone else!");
       }
   });
```

```
class RoboDog extends Robot
        implements Pet
{
   private String name;

   public RoboDog() {
      this("Noname Robot");
   }

   public RoboDog(String name) {
      this.name = name;
      Dog.pets.add(this);
   }

   public String getName() {
      return name;
   }

   public void beFriendly() {
      System.out.println("I'm friendly!");
   }
}
```

Using of inner and anonymous classes

```
// create anonymous class inherited from Cat
pets.add(new Cat("Tiger Tigra") {
    public void beFriendly() {
        System.out.println(
                        "I'm not friendly! "+
                        "I'm a Tiger!");
    }
    public String getName() {
        return "";
    }
});

// adding Pet interface implementation
pets.add(new Pet() {
    public void beFriendly() {
    }
    public String getName() {
        return "I'm a Pet";
    }
});

// ask all pets to be friendly
for (Pet p: pets) {
    p.beFriendly();
}
```

```
class Cat implements Pet {
    public String name;
    public Cat(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void beFriendly() {
        System.out.println("I'm friendly!");
    }
}
```

```
interface Pet {
    public String getName();
    void beFriendly();
}
```

Exercise

Lab guide:

- Exercise 13