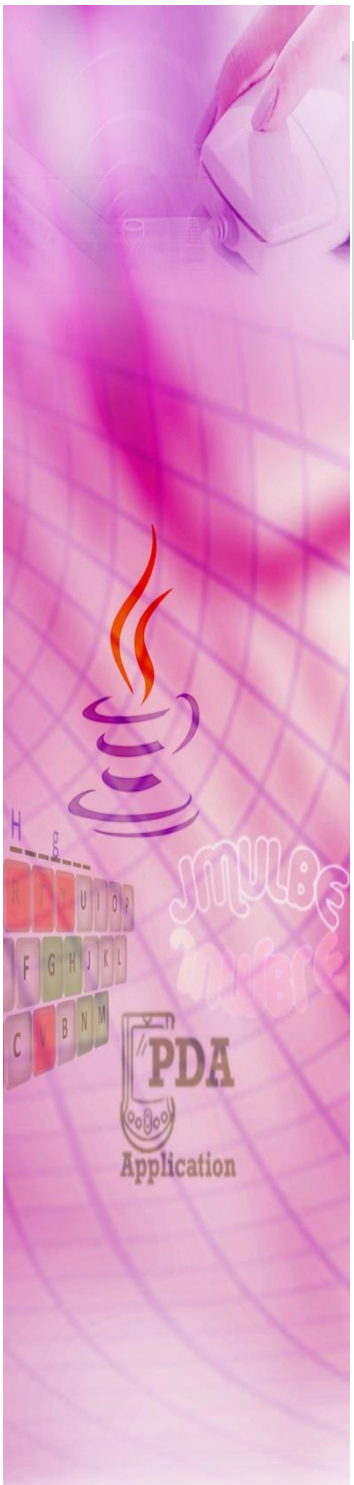


Instructor Inputs

Session 1

Notes for Faculty

The faculty needs to consider the following points for this course:

- The study material for this course is divided into three books, Programming in Java – Student Guide (SG), Activity Book, and Toolbook. The SG contains the concepts, and the Activity Book provides the activities to help the learners understand how to implement these concepts. The Toolbook provides instructions to install and use the software required for this course. The SG contains the placeholders for the activities.
- While going through the SG, whenever you find the placeholder for an activity or a task, you need to refer to the Activity Book to see the steps for performing the activity.
- For conducting the classroom sessions, you need to refer to the SG, Toolbook, and the Activity Book.
- For conducting the machine room session, you need to refer to the Activity Book that lists the exercises to be performed in these sessions.
- While developing the activities, exercises, and additional exercises, the implementation of concepts taught in the SG are kept in focus. However, the logic to perform validation of inputs is not implemented. Therefore, if students want, they can add the code to implement validations of inputs.

Session Overview

This session covers Chapter 1 and the following topics of the Processing Strings Using Regex section of Chapter 2 of the book, Programming in Java – SG:

- Working with the Pattern and Matcher Classes
- Working with Character Classes

In this session, the students will learn the basic concepts of creating inner classes and implementing type casting. In addition, the students will learn the basic concepts of the `Pattern`, `Matcher`, and `Character` classes.

Creating Inner Class

Handling Tips

Start the session with the students by discussing the scenario of the Java application that processes the credit card payments for clients. Thereafter, explain the importance of the inner class in the programming language. Further, list the different types of inner class. While explaining the regular inner class, you can notify the students that private fields are accessible to the inner class. In addition, demonstrate the code given under the Regular Inner Class subtopic in the Additional Examples topic. Further, while explaining the static inner class, emphasize on the non static variables that are not accessible inside the static class. In addition, demonstrate the code given under the Static Inner Class subtopic in the Additional Examples topic. Thereafter, explain the method-local inner class that can access both, the member variables and the class variables. In addition, explain the code given in the Additional Examples topic. While explaining the anonymous inner class, emphasize on the anonymous inner class that can be created for an existing class or interface. In addition, explain the code given in the Additional Examples topic.

Additional Examples

Regular Inner Class

Consider the following code of the regular inner class:

```
class MyOuter
{
   private int x = 7;
   public void makeInner()
   {
          MyInner in = new MyInner();
          in.seeOuter();
   }
   class MyInner
   {
          public void seeOuter()
          {
                 int x=8;
                 System.out.println("Inner x is " + x);
                 System.out.println("Outer class x " + MyOuter.this.x);
          }
```

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

The preceding code generates the following output:

```
Inner x is 8
Outer class x 7
```

Static Inner Class

Consider the following code of the static inner class:

```
class MyOuter
{
    static class Nest
    {

        void go()
        {
                int x=10;
                System.out.println("value of x:"+x);
        }
    }
}

class MyInner
{
    static class B2
    {
        void goB2()
        {
                int x=20;
                System.out.println("Value of inner class x is:"+x);
        }
    }

    public static void main(String[] args)
    {
        MyOuter.Nest n = new MyOuter.Nest();
        n.go();
        B2 b2 = new B2();
        b2.goB2();
    }
}
```

The preceding code generates the following output:

```
value of x:10
value of inner class x is:20
```

Method-local Inner Class

Consider the following code of the method-local inner class:

```
class MethodLocal
{
    private int x = 30;
    void display()
    {
        final int y = 50;
        class Local
        {
            void msg()
            {
                System.out.println("Value of x is:"+ x);
                System.out.println("Value of y is:" +y);
            }
        }
        Local l = new Local();
        l.msg();
    }
    public static void main(String args[])
    {
        MethodLocal obj = new MethodLocal();
        obj.display();
    }
}
```

The preceding code generates the following output:

```
Value of x is:30
Value of y is:50
```

Anonymous Inner Class

Consider the following code that implements the anonymous inner class:

```
    interface HelloWorld {
        public void greet();
        public void greetSomeone(String someone);
    }
public class HelloWorldAnonymousClasses {

    public void sayHello() {

        class EnglishGreeting implements HelloWorld {
            String name = "world";
            public void greet() {
                greetSomeone("world");
            }
            public void greetSomeone(String someone) {
                name = someone;
                System.out.println("Hello " + name);
            }
        }
```

```
        HelloWorld englishGreeting = new EnglishGreeting();

        HelloWorld frenchGreeting = new HelloWorld() {
            String name = "tout le monde";
            public void greet() {
                greetSomeone("tout le monde");
            }
            public void greetSomeone(String someone) {
                name = someone;
                System.out.println("Salut " + name);
            }
        };

        HelloWorld spanishGreeting = new HelloWorld() {
            String name = "mundo";
            public void greet() {
                greetSomeone("mundo");
            }
            public void greetSomeone(String someone) {
                name = someone;
                System.out.println("Hola, " + name);
            }
        };
        englishGreeting.greet();
        frenchGreeting.greetSomeone("Fred");
        spanishGreeting.greet();
    }

    public static void main(String... args) {
        HelloWorldAnonymousClasses myApp =
            new HelloWorldAnonymousClasses();
        myApp.sayHello();
    }
}
```

The preceding code generates the following output:

```
Hello world
Salut Fred
Hola, mundo
```

Activity 1.1: Creating Inner Classes

Handling Tips

Discuss the problem statement with the students.

The solution file, **Candidates.java**, for this activity is provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 01\Activity 1.1\Solution**

Implementing Type Casting

Handling Tips

Discuss the scenario of the employee class that has two child classes, Manager and Supervisor, and explain the need of type casting to the students. Next, tell the students that Java supports type casting of primitive data types and objects. Then, explain the type casting of primitive data types with the code snippets given in the SG. In addition, demonstrate the example given under the Type Casting Primitive Data Types subtopic in the Additional Examples topic. Further, while explaining the type casting object, demonstrate the example given under the Type Casting Object subtopic in the Additional Examples topic.

Additional Examples

Type Casting Primitive Data Types

The following code demonstrates type casting of the primitive data types:

```
public class Casting
{
    public static void main(String args[])
    {
        System.out.println("(int) 7.9="+7.9 );
        System.out.println("(double) 7.9="+(double)7.9 );
        System.out.println("(int) 7.9+(int) 6.3 ="+((int) 7.9+(int) 6.3) );
        System.out.println("((double) 7.9+(double) 6.3) ="+((double)
7.9+(double)     6.3) );
        System.out.println("(double) 15/2="+(double) 15/2);

    }
}
```

The preceding code generates the following output:

```
(int) 7.9=7
(double) 7.9=7.9
(int) 7.9+(int) 6.3 =13
((double) 7.9+(double) 6.3) =14.2
(int) 15/2=7
(double) 15/2=7.5
```

Type Casting Object

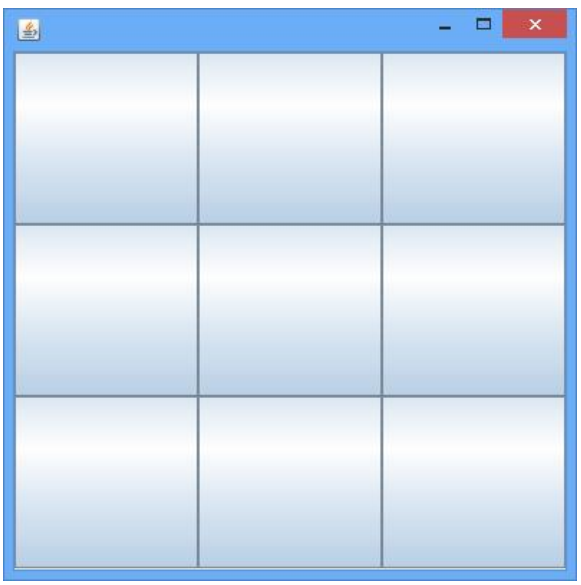
Consider the following code that demonstrates the use of downcasting:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
class GridFrame extends JFrame implements ActionListener{
    JButton button[]=new JButton[10];
    GridFrame(){
        setLayout(new GridLayout(3,3));
        for(int i=1;i<button.length;i++){
```



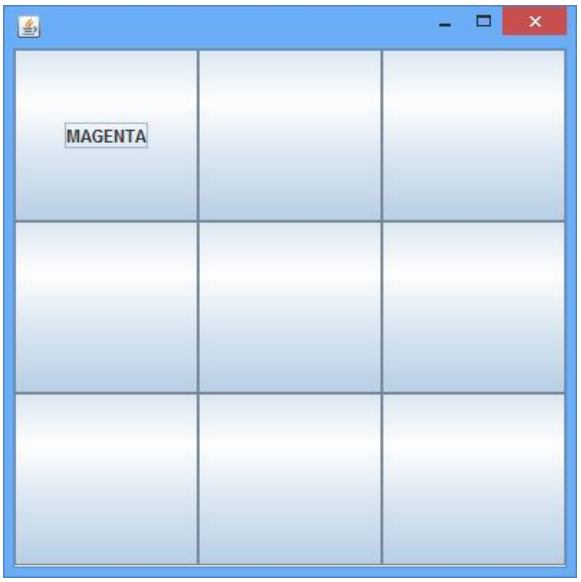
```
            button[i]=new JButton();
            add(button[i]);
            button[i].addActionListener(this);
        }
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400,400);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae){
        String
colors[]={"RED","GREEN","BLACK","BLUE","PINK","WHITE","CYAN","MAGENTA","ORANG
E","YELLOW"};
        Random r=new Random();
        int i=r.nextInt(10);
        JButton b=(JButton)ae.getSource();
        b.setText(colors[i]);
    }
}
public class DowncastingDemo {
    public static void main(String args[]){
    GridFrame g=new GridFrame();
    }
}
```

The preceding code generates the following output:



The Output

Once you click the first button, the following output is displayed:



The Output

In the preceding code, the `getSource()` method of the `ActionEvent` class is used to retrieve the component, which was clicked by the user. However, the return type of the `getSource()` method is of the `Object` class. Therefore, the value returned by the `getSource()` method is downcasted to the object of the `JButton` class.

Note

The output may vary based on the random number generated.

Processing Strings Using Regex

Handling Tips

Discuss the scenario of the Java application that provides the functionality to search the text, hello, in a file. Briefly discuss the concepts of regular expression. Tell the students about the regular expression. While discussing the methods of the `Pattern` and `Matcher` classes, explain the methods given under the Additional Inputs topic. Further, while discussing the `character` class constructs, explain the code given in the Additional Inputs topic.

Additional Inputs

Working with the Pattern and Matcher Classes

The following table lists the methods of the `Pattern` class.

Method	***Description***

`static String quote(String s)`	*Is used to return a pattern* `String` *for the specified* `String s`*. This method produces a* `String` *that can be used to create a* `Pattern` *that would match the string,* `s`*, as if it were a literal pattern.*
	Consider the following code snippet:
	`String pattern=Pattern.quote("^[abc]");`
	`System.out.println(pattern);`
	Output:
	`\Q^[abc]\E`

`int flags()`	*The* `Pattern` *class contains a list of flags that you can use to make the pattern matching behave in certain ways. For example, in the statement,* `Pattern pattern = Pattern.compile(patternString, Pattern.CASE_INSENSITIVE);`*, the* `Pattern.CASE_INSENITIVE` *flag makes the pattern matching by ignoring the case of the text.*

The Methods of the Pattern Class

The following table lists the methods of the `Matcher` class.

Method	***Description***
`String group()`	*Is used to return the input subsequence matched by the previous match.*
`Matcher reset()`	*Is used to reset the matcher.*
`public boolean lookingAt()`	*Is used to match the input sequence, starting at the beginning of the region, against the pattern.*

The Methods of the Matcher Class

Working with Character Classes

Consider the following code of the character class:

```
import java.util.regex.*;
import java.util.*;
public class PatternMethods
{
    public static void main(String args[])
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the desired pattern: ");
        Str      ing pattern = input.nextLine();
        System.ou       t.println("Enter the text: ");
        String matcher = input.nextLine();
        Pattern myPattern2 = Pattern.compile(pattern);
        Matcher myMatcher2 = myPattern2.matcher(matcher);
        Boolean myBoolean2 = myMatcher2.matches();
        boolean b = myBoolean2;
        if (b == true)
        {

           System.out.println("I found the text:" + myMatcher2.group() + "\n"
                    + "Starting at:" + myMatcher2.start() + "\n"
                    + "Ending at index:" + myMatcher2.end());
        }
        else if (b == false)
        {
            System.out.println("No match found");
        }
    }
}
```

While executing the preceding code, if you provide the pattern as `[^abc]at` and matcher as `cat`, the following output will be displayed:

```
No match found
```

In the preceding output, the overall match does not succeed as the first letter does not match one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[^abc]at` and matcher as `rat`, the following output will be displayed:

```
I found the text:rat
Starting at:0
Ending at index:3
```

In the preceding output, the overall match succeeds as the first letter matches one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[a-c]at` and matcher as `rat`, the following output will be displayed:

```
No match found
```

In the preceding output, the overall match does not succeed as the first letter does not match one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[a-c]at` and matcher as `bat`, the following output will be displayed:

```
I found the text:bat
Starting at:0
Ending at index:3
```

In the preceding output, the overall match succeeds as the first letter matches one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[a-b[e-f]]at` and matcher as `cat`, the following output will be displayed:

```
No match found
```

In the preceding output, the overall match does not succeed as the first letter does not match one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[a-f&&[d-f]]at` and matcher as `fat`, the following output will be displayed:

```
I found the text:fat
Starting at:0
Ending at index:3
```

In the preceding output, the overall match succeeds as the first letter matches one of the characters defined by the character class.

While executing the preceding code, if you provide the pattern as `[a-f&&[d-f]]at` and matcher as `cat`, the following output will be displayed:

```
No match found
```

In the preceding output, the overall match does not succeed as the first letter does not match one of the characters defined by the character class.

FAQs

- *Are there some rules defined for type casting in Java?*

  Ans: Yes, the rules defined for type casting in Java are:

  If the range is narrowed, an explicit type cast is required.

  An implicit cast occurs in the following orders:

  - byte → short → int → long → float → double
  - char → int → long → float → double

- *Can we have an inner class inside an inner class?*
  Ans: Yes, we can have inner class inside an inner class.

- *Which modifiers can be applied to the method-local inner class?*
  Ans: Only the abstract or final keyword is allowed.

- *What are the advantages of an inner class?*
  Ans: Advantages of inner class are:
  - It is a way of logically grouping classes that are only used in one place.
  - It increases encapsulation.
  - Nested classes can lead to more readable and maintainable code.

- *Is there any method available in the* `Matcher` *class to find the number of occurrence of a particular word?*
  Ans: No, there is no such method available in the `Matcher` class.