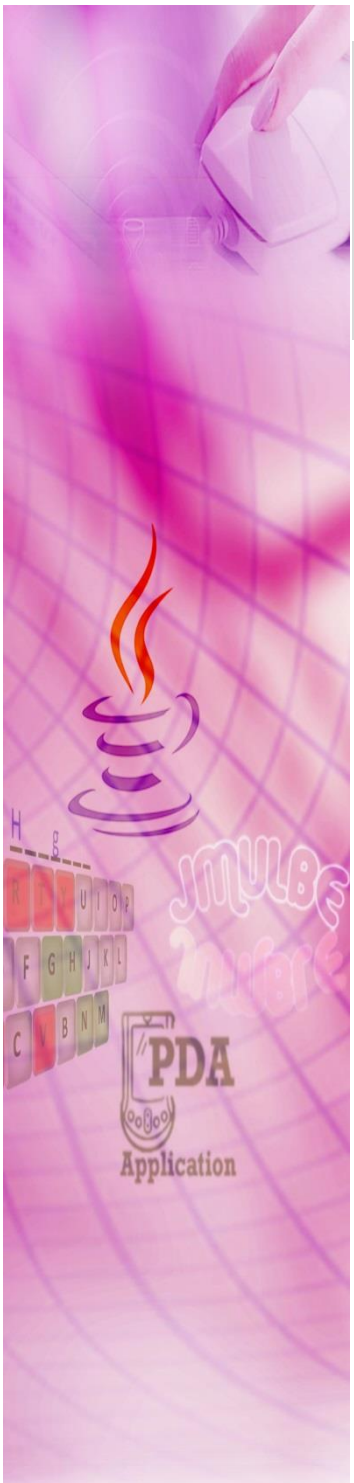

Instructor Inputs

Session 2

Session Overview

This session covers the Working with Quantifiers topic of the Processing Strings Using Regex section, the Implementing Localization section of Chapter 2, and the Creating User-defined Generic Classes and Methods section of Chapter 3 of the book, Programming in Java – SG.

In this session, the students will learn about quantifiers and localization. In addition, this session will help the students to become familiar with the user-defined generic classes and methods.

Processing Strings Using Regex

Handling Tips

Start the discussion by explaining the scenario where you need to specify the number of times a character or a sequence of character appears in the expression. Next, list the different types of quantifier and explain the difference between these quantifiers. In addition, explain the code given in the SG. Further, while explaining this topic, you can discuss the code given in the Additional Examples topic.

Additional Examples

Consider the following code that checks whether the word, hello, has occurred thrice or not:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

class CaptureGroups
{
    public static void main(String args[])
    {
            Pattern mypattern=Pattern.compile("(hello){3}");
            Matcher mymatcher=mypattern.matcher("hellohellohello");
            Boolean MyBoolean=mymatcher.matches();
            System.out.println(MyBoolean);

    }
}
```

Once the preceding code is executed, the following output is displayed:

```
true
```

In the preceding code, `MyBoolean` will have the value, `true`, only when the `hello` word has occurred three times.

Activity 2.1: Processing Strings Using Regex

Handling Tips

Discuss the problem statement with the students.

To perform the activity, 2.1, you need to use the **LoginForm.txt** file, which is provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 02\Activity 2.1\Input Files**

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

- **Datafiles For Faculty\Activities\Chapter 02\Activity 2.1\Solution**

Implementing Localization

Handling Tips

Start discussing the scenario of the hotel management application and explain the need for localization. Next, tell the students that to implement localization, Java provides a set of predefined language and country codes. In addition, to work with the language and country codes, Java provides the `java.util.Locale` class. While explaining localizing date, notify the students that the `java.text.DateFormat` class is used to format the date according to the locale. In addition, while discussing this topic, you can discuss the language codes and country codes given in the Additional Inputs topic. Further, demonstrate the code given under the Localizing Date subtopic in the Additional Examples topic. Thereafter, while explaining localizing currency, notify the students that the `java.text.NumberFormat` class is used to format the currency according to the locale. Next, while explaining the localizing text to the students, emphasize on the resource bundle creation. Further, demonstrate the code given under the Localizing Text subtopic in the Additional Examples topic.

Additional Inputs

The following table lists some of the commonly-used language and country codes.

Language	***Country***	***Language Code***	***Country Code***
Arabic	*Algeria*	`ar`	`DZ`
Arabic	*Bahrain*	`ar`	`BH`
Arabic	*Egypt*	`ar`	`EG`
Chinese (Simplified)	*China*	`zh`	`CN`
Chinese (Traditional)	*Hong Kong*	`zh`	`HK`
Danish	*Denmark*	`da`	`DK`
English	*Australia*	`en`	`AU`

Language	*Country*	*Language Code*	*Country Code*
English	*India*	*en*	*IN*
French	*Canada*	*fr*	*CA*
German	*Germany*	*de*	*DE*
Spanish	*Argentina*	*es*	*AR*
Turkish	*Turkey*	*tr*	*TR*

The Commonly-used Language and Country Codes

Additional Examples

Localizing Date

Consider the following code that displays the date in different formats, such as MEDIUM, FULL, LONG, and SHORT:

```
import java.util.*;
import java.text.*;
public class DateFormats {
   public static void main(String[] args) {
      Date dt = new Date();

      DateFormat[] dtformat = new DateFormat[6];
      dtformat[0] = DateFormat.getInstance();
      dtformat[1] = DateFormat.getDateInstance();
      dtformat[2] = DateFormat.getDateInstance(DateFormat.MEDIUM);
      dtformat[3] = DateFormat.getDateInstance(DateFormat.FULL);
      dtformat[4] = DateFormat.getDateInstance(DateFormat.LONG);
      dtformat[5] = DateFormat.getDateInstance(DateFormat.SHORT);

      for(DateFormat dateform : dtformat)
         System.out.println(dateform.format(dt));
  }
}
```

Once the preceding code is executed, the following output is displayed:

```
6/30/13 4:31 PM
Jun 30, 2013
Jun 30, 2013
Sunday, June 30, 2013
June 30, 2013
6/30/13
```

In the preceding code, the overloaded static method, `getDateInstance()`, of the `DateFormats` class is used to create the various instances of the `DateFormats` class with different date formats. In addition, the `format()` method of the `DateFormats` class is used to format the current date according to the type of instance created.

Localizing Text

Consider the following code that localizes the text by using `ListResourceBundle`:

```
import java.util.*;

class MyResources extends ListResourceBundle
{
    protected Object[][] getContents()
    {
        return new Object[][]{
                    {"hello", "你好"},
                    {"bye", "再见"},
                    {"goodnight", "晚安"}
                };
    }
}
class ListResourceBundleDemo
{
    public static void main(String[] args)
    {
        MyResources mr = new MyResources();
        System.out.println("" + mr.getString("hello"));
        System.out.println("" + mr.getString("bye"));
        System.out.println("" + mr.getString("goodnight"));
    }
}
```

In the preceding code, the `MyResources` class that extends the `ListResourceBundle` class has been created. Inside the class, a two-dimensional array is initialized with the key-value pairs. The keys are the first element in each pair, such as hello, bye, and goodnight. The values are the second element in each pair. The `getString()` method is used to get a string for the given key from the list resource bundle.

Activity 2.2: Implementing Localization

Handling Tips

Discuss the problem statement with the students.

To perform the activity, 2.2, you need to use the **GameWindow.txt** and **Puzzle.txt** file, which are provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 02\Activity 2.2\Input Files**

The solution files, **GameWindow.java** and **Puzzle.java**, for this activity are provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 02\Activity 2.2\Solution**

Creating User-defined Generic Classes and Methods

Handling Tips

Start explaining to the students that the class that can return various objects can be created by using the generics feature of Java. Then, explain a generic class and how the generic classes can be created by using the type parameter. Next, explain the type parameter with the help of the code given in the SG. In addition, while discussing this topic, you can discuss the information given under the Creating Generic Classes subtopic in the Additional Inputs topic. Thereafter, discuss the generic methods and how the generic methods can be created inside a class. Next, emphasize on the point that the generic method can be created inside the generic and non generic classes. In addition, while discussing this topic, you can discuss the information given under the Creating Generic Method subtopic in the Additional Inputs topic.

Additional Inputs

Creating Generic Classes

In addition to generic classes, you can create generic interfaces. Generic interfaces are interfaces with the type parameter. The following code snippet shows how to create the generic interface:

```
interface GenInt<T>
{
    public void setValue(T t);
    public T getValue();
}
class GenIntDemo implements GenInt<Integer>
{

    public int t;
    public void setValue(Integer t)
    {
        this.t = t;
    }

    public Integer getValue()
    {
        return this.t;
    }

}

public class TestClass
{
    public static void main(String[] args)
    {
        GenIntDemo obj = new GenIntDemo();
        obj.setValue(25);
        System.out.println(obj.getValue());
    }
}
```

In the preceding code, the `GenInt` interface is created, which is a generic interface. In addition, the `GenIntDemo` class implements the `GenInt` interface by passing `Integer` as the type parameter.

Creating Generic Method

In addition to the generic method, you can create the generic constructor. Both, the generic class and non generic class, can contain a generic constructor.

The following code snippet shows how to create the generic constructor:

```
public class GenConClass
{
    private int value;

    <T extends Number> GenConClass(T v)
    {
        value = v.intValue();
    }

    public void display()
    {
        System.out.println(value);
    }

    public static void main(String[] args)
    {
        GenConClass obj = new GenConClass("Hello");   //Line 1
        obj.display();
        GenConClass obj2 = new GenConClass(25);       //Line 2
        obj2.display();
    }

}
```

In the preceding code, the `GenConClass` class contains the generic constructor that can accept the object of the type, `Number`, only. In addition, `line 1` will generate a compilation error because `Hello` is the `String` object that cannot be passed to the generic constructor. On the other hand, `line 2` will be compiled successfully because `25` is the `Integer` object that can be passed to the generic constructor.

FAQs

- *Is localization of text possible without resource bundle?*

  Ans: Yes, localization of text is possible by using the `ListResourceBundle` class. It is a convenience class provided by Java.

- *Is it possible to change the locale at runtime?*

  Ans: Yes, it is possible to change the locale at runtime.

- *Can a generic array be created?*

  Ans: No, a generic array cannot be created because the compiler cannot identify the type of elements for array.

- *Can generics be used with the primitive data types?*

  Ans: No, the primitive data types cannot be used with generics because generics only work with the objects. However, to work with the primitive data types, you can use the wrapper classes to encapsulate the primitive type.

- *Can a type parameter be instantiated?*

  Ans: No, creating an instance of a type parameter is not possible because the type parameter does not exist at runtime. The main reason behind this is that JVM does not know what type of object needs to be created.

- *Can a class' static field be of the generic type?*

  Ans: No, a class' static field cannot be of the generic type because a static field is shared by all non static objects as a static field does not work with a single object.