

## UNIT III

### Event Handling :

Event is defined as change of status. When a graphical component for example like a button clicked, then the status of the button is changed. When an item in the List or Choice object is selected or deselected, then there is a change of status occurred. The java provides a facility of doing some process when an event is raised called event handling. In all RAD tools (Rapid Application Development) tools (i.e. GUI Design tools) like java, VC++, VB.Net etc the event handling is classified into two models. Event association model and event delegation model. The java supports event delegation model.

Event Delegation Model : in this model, when an event is raised, the program control is delegated (handed over controls with) to a specific method. Irrespective of component name. for example, when a button clicked, the program control is delegated to a method with name actionPerformed() irrespective of button name.

To implement event delegation model, the java.awt.event package has number of classes and interfaces.

Event Classes : the event classes are used as carriers of source component name from the UI into the program, so that in the code, the programmer can check for the source component name, to develop different processes for different components. These event classes are suffixed with a term Event in their names.

Event Listeners : the java.awt.event package has number of interfaces that are identified with a suffix term "Listener" in their names. The java says, implement appropriate listener, to handle a specific event, as the corresponding Listener interface has appropriate method declared with in, and once programmer implements that Listener, he must override the interface's method, which is being called by the JRE when event is raised.

The programmer need to implement appropriate listener interface, and override corresponding method, and place the code that need to be executed as definition of overridden method. For making a component to raise event, the component must be registered with appropriate listener with the method addTYPEListener(TYPEListener instance).

When an event is raised, there are two processes performed internally at a time, called the internal mechanism. The appropriate event class is instantiated and the source component name is stored into that instance and the corresponding overridden method is called by JRE and the Event class instance is passed as argument to the overridden method.

The different classes and interfaces related with the event handling mechanism are :

| <u>Component</u> | <u>Event name/<br/>Class name</u> | <u>Listener Interface</u>             |
|------------------|-----------------------------------|---------------------------------------|
| Button           | ActionEvent                       | ActionListener                        |
| TextField        | TextEvent                         | TextListener                          |
| List/ Choice     | ItemEvent                         | ItemListener                          |
| Mouse            | MouseEvent                        | MouseListener/<br>MouseMotionListener |
| Key Board        | Key Event                         | KeyListener                           |

----

Q. What is Event Handling

Q. Explain event handling mechanism in java

Q. Explain event delegation model

Q. Explain different classes / interfaces related with event handling

-----

ActionEvent and ItemEvent:

The ActionEvent is raised.....

When a button clicked

When user hits ENTER in a text field object

When user double clicks an item in List

To handle ActionEvents, the class must implement ActionListener interface and override its method public void actionPerformed(ActionEvent), and place the process as definition of this method.

For raising event, we generally implement ActionListener interface by our class and register the component with a method call addActionListener(this). Here the this refers the current object that implemented ActionListener interface.

For example, when a button clicked, the JRE creates `ActionEvent` class instance and stores the button name that raised event (like `b1` or `b2` etc), and the JRE calls the method `actionPerformed()` and passes this event instance as argument. so thus when the method is overridden, the `actionPerformed()` receives an argument of `ActionEvent`. The `ActionEvent` has a method `getSource()` that returns the component name, that can be further used to develop different processes for different components.

The `ItemEvent` is raised.....

When user selects an item in `Choice`

When single clicks an item in `List` object

When checks/ unchecks a `Checkbox` / radio button

For handling the item events, we implement `ItemListener` interface and register the component with the listener with `addItemListener()` and override the method `itemStateChanged()`.

----

Q. Explain Action and Item Events.

Q. Write about handling events with different components in java.

-----

Adapter Classes:

The interfaces when implemented, the class must override all the methods of the corresponding interface. When handling events with mouse , keyboard, frame etc the corresponding interfaces have more than one method to be overridden. Many a times, this leads to a situation where programmer needs to override methods that are not required for application, which leads to a situation of unnecessary overriding of methods.

To reduce this unnecessary overriding of methods, the `java.awt.event` package has some classes called adapter classes that adapt features of their corresponding listener.

These adapter classes are abstract classes, that implement their corresponding listeners and override all the methods of that listener interface with empty definitions. When programmer needs limited events are to be handled and the listener interface has more methods that are not required in that situation, then

instead of registering component with interface instance, programmer can register with corresponding adapter class, overriding only required method.

Ex:

```
addWindowListener ( new WindowAdapter() {  
  
    public void windowClosing (WindowEvent e)  
    {  
        System.exit(0);  
    }  
});
```

The interfaces that have only single method to be overridden like ActionListener, ItemListener, TextListener donot have adapters.

-----

Q. Write about adapter classes.

Q. How to reduce unnecessary overriding of methods in event handling?

-----

### **Exception Handling – Definition**

There are three types of errors in java

- 1.Compile-time errors
- 2.Run time errors
- 3.logical errors

#### **COMPILE TIME ERRORS:**

These are errors, which prevents the code from compiling because of error in the syntax such as missing a semicolon at the end of a statement or due to missing braces, class not found, etc. These errors will be detected by java compiler and displays the error onto the screen while compiling.

Ex:

```
System.out.println(“ HELLO WORLD); // ” missing etc
```

## RUN TIME ERROR:

These errors are errors which occur when the program is running. Run time errors are not detected by the java compiler. It is the JVM which detects it while the program is running.

Ex:

```
int a[]=new int[5];
```

```
System.out.println(a[5]); //the 4 is the last index. It is expressed as an exception by the JVM.
```

## LOGICAL ERRORS:

These errors are due to the mistakes made by the programmer. It will not be detected by a compiler nor by the JVM. Errors may be due to wrong idea or concept used by a programmer while coding or by giving wrong data given input by user.

Ex:

```
inta,b,c;
```

```
a=sc.nextInt();
```

```
b=sc.nextInt();
```

```
c=a/b; //here if the user gives input of 0 as b value, then the JRE could not execute code and raises error by throwing an exception
```

The logical errors and runtime errors when raised in program, the JRE could not execute the code and terminates abruptly the function in which it could not executed code, and this case is said that the JRE has thrown an Exception. The java provides a software mechanism called Exception Handling to let the programmer to control program execution, such that it won't terminate abruptly. The exception handling is supported with keywords try-catch-finally, throw and throws.

The main benefit of exception handling is to debug the program, such that the program can be executed as robust program.

---

Q. Give an introduction of exceptions and benefits of exception handling.

---

Types of Exceptions :

Classification of exceptions, exception hierarchy, checked exceptions and unchecked exceptions:

The java exceptions are classified into two types.

- Checked exceptions / compile time exceptions
- Unchecked exceptions / runtime exceptions

Checked exceptions: The code in java that has hardware interactivity or interactivity of other runtime environments always must be handled to achieve more perfection (Robustness). To make sure programmer to use appropriate handler, these exceptions are declared as checked / compile time exceptions. Unless these are handled, the code will not be compiled.

Runtime Exceptions: the exceptions that are thrown by JRE during running of program, based on data given input by user or the logic implemented in program are called runtime exceptions.

The java can handle more than 600 types of pre-defined exceptions and all these are derived from java.lang.Exception class.

The different pre-defined exceptions are:

Compile time exceptions / checked exceptions:-

IOException : when any IO operation is performed

FileNotFoundException: when file processing code is written

SQLException: when database connectivity established

InterruptedException: when the sleep() of thread class is used

ClassNotFoundException: when a class is loaded into memory by program for further use etc situations

Runtime exceptions / unchecked exceptions

ArrayIndexOutOfBoundsException :when we mention the index  $\geq$  length of array

StringIndexOutOfBoundsException : when we use the index  $\geq$  length() of String

ArithmeticException :when division is done with ZERO

NumberFormatException : when a non-numeric value is tried to be parsed

NullPointerException : when a class reference is used prior its instantiation  
(allocation of memory

---

Q. Write about exception handling in java

Q. explain different types of exceptions in java

---

Usage of try-catch-finally, throw and throws.

The java provides a software mechanism called Exception Handling to let the programmer to control program execution, such that it won't terminate abruptly. The exception handling is supported with keywords try-catch-finally, throw and throws.

try-catch-finally: this block is used to handle the code, such that when an exception is thrown, the JRE customizes the process to be done without terminating abruptly.

Syntax :

```
try{
    code expected to throw exception
}
catch(TYPEException e){
    Code that need to be executed when exception is thrown and handled
}
finally{
    default block
}
```

Here the TYPEException in catch() is replaced with appropriate exception type.

We place the code that is expected to throw exception in try block. A try must have at least one catch() or finally. But a try can have any number of catches, each handling a specific type of exception.

Ex:

```
try{
    int x=Integer.parseInt(sc.nextLine());
    int y=Integer.parseInt(sc.nextLine());
    int z=x/y;
    System.out.println(z);
} catch(ArithmeticExceptionae){
    System.out.println("don't enter zeros");
} Catch(NumberFormtException ne){
    System.out.println("enter only numbers");
}
```

The finally is default block that executes in all the cases. Generally, the closing of streams, database connections etc is one with in finally block.

---

Q. Write about exception handling with try-catch-finally

---

throw, throws keywords :

The “throw” is a keyword, used to throw an exception explicitly. The “throw” can throw a pre-defined exception or any derived class from Exception including user defined exceptions.

Ex:

```
throw new ArithmeticException();
```

The throw must be placed within a try block to handle it.

The “throws” is a keyword used along with method signature to specify that the method throws a specific exception and the programmer must provide proper handler. i.e. it is used to specify that the programmer must handle that exception with catch(), unless which the code will not be compiled.

Ex:

abc.java

```
class abc {  
    public int divideData( int x, int y) throws ArithmeticException { //line 2  
        return x/y;  
    }  
}
```

-----

bca.java

```
class bca {  
    public static void main(String args[]){  
        abc a=new abc();  
        try{  
            System.out.println(a.divideData(5,0) ); //line 5  
        } catch(ArithmeticException e){  
            System.out.println(e);  
        }  
    }  
}
```

If the line at line 5 is not placed under try block, then the program will not be compiled. Thus, the “throws” at “line 2” makes sure when the add() is called, it must be tried and caught. If the programmer doesn’t have anything to do in catch(), then he may declare that the calling method also “throws” the exception, without handling.

Ex:

```
bca.java
class bca {
    public static void main(String args[]) throws ArithmeticException{
        abc a=new abc();
        System.out.println(a.divideData(5,0) );
    }
}
```

---

Q. Write about throw and throws clauses in java

---

Packages – Creating Packages, Importing Packages :

Packages:

When java is installed into system, there will be two components installed into system viz., JVM and Java API. The JVM is java virtual machine and the API is Application Programming Interface, simply java library.

The java API includes no. of pre-defined classes, grouped based on their purpose and placed into folders called packages. A package in Java is used to group related classes. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories.

Built-in Packages

User-defined Packages (create your own packages)

The main package in API is called “java” and it has no. of sub packages, where each package has no. of classes related for a specific purpose. These packages must be linked to our program (as the libraries are linked in C with #include ) with the keyword import.

Syntax of import :

import package\_name.Class\_Name; imports specific class into our program.

Ex: import java.util.Scanner; imports Scanner class of util package into our program.

import Package\_name.\*; imports all classes in that package

Ex: `import java.util.*;` imports all classes of util package into our program.

There are many system defined packages like:

lang: this package has classes to handle basic features of java LANGUAGE.

util: has general utilities like Scanner, Vector, ArrayList, Arrays, Stack, LinkedList

etc classes.

Sql: this package has classes for database connectivity.

applet: for applet programming

io: has classes for managing input / output streams etc

to make use of the classes, we must import the corresponding package into our program.

User defined packages: we can also create our packages and organize our classes into them. To create a user defined package, we must place a “package” statement in first line of the program.

Ex: `package mypack;`

When we need to place a class / interface into a package, then the program must have a package statement, the class must be declared as public class, and we must compile that class by using the option `-d` with javac command, specifying where to place the .class file.

Ex:

```
packagemypack;
```

```
public class MyClass{  
    public int add(int a, int b) { return a+b; }  
}
```

While compiling this class we must use the option `-d` with javac command.

Syntax: `javac -d <path> FileName.java`

Here the <path> specifies where to place generated class files/ where to create the package.

Ex: `javac -d c:\ MyClass.java`

For using this class, we must import the package into program and make use of the class.

Import `mypack.MyClass`;

```
classMyAddition{
    public static void main(String a[]){
        MyClass m=new MyClass();
        System.out.println(m.add(5,6) );
    }
}
```

We use the option `-classpath` with `javac` command while compiling this class.

Syntax :`javac -classpath<path> FileName.java`

Here path specifies where to file class files / package

Ex:

`Javac -classpath c:\ MyAddition.java`

While running this program, we use the option `-cp` with `java` command specifying the location of both class files, i.e class of the package and also current folder, where the current folder is specified with a “.” (DOT).

Syntax : `java -cp<paths>ClassName`

Ex: `java -cp .; c:\ MyAddition`

Once a package is created, we can add any number of classes to it. But all the classes must be public and saved with classname. If a class placed within a package is not public, then that class cannot be used outside of the package and they are called hidden classes.

Static import: we can access the static members of the class with class name. But when we need to use many of the class static members, then we can import all static features of the class into our program with a static import. Then we need not use class name to access the static features.

If the class has all static members, then instead of a general import statement, we can have a static import statement that imports all static features into class that can be accessed directly without class name.

Ex:

```
import static java.lang.Math.*;
```

```
class abc{  
    public static void main(String a[]){  
        System.out.println(max(5,6));  
    }  
}
```

---

Q. Write about packages in java

Q. How to create user defined packages and use them

Q. What is the purpose of packages and how to use them

---

Access Protection : in java, there are different access specifiers and modifiers used for accessibility control, like private, protected, public and default access specifiers used for data controls and security.

In java, the top level class cannot be private, as we cannot use it directly. An inner class can be private, but it can be used only within the class in which it is declared.

The protected are used for being accessed within the inheritance hierarchy.

Generally the classes are declared with public accessibility, so that they can be used either within that package or outside of the package.

If no access specifier mentioned, the java's default accessibility is applied, which has a specification of being accessed within current package. (not outside of the package).

If a class is declared with public accessibility within a package, then it can be accessed outside of the package also. If no access specifier mentioned while declaring the class, then the default accessibility is applied, where that class can be used in the classes within that package and cannot be accessed outside of the class, and such classes are called hidden classes.

If a feature declared as protected within a class, then only it is accessible in derived classes that are outside of that package.

-----

Q Explain accessibility options of features of a package.

-----