Unit 3

UNIT 3 - Contents

- Arrays
- Strings
- Packages
- Creating Packages
- Using Packages
- Visibility control
- Exception Handling Techniques
- try-catch
- throw and throws
- finally
- Multithreading
- Creation of Multi-threaded programs
- Thread Class
- Runnable interface

Arrays

- A group of related data items that share a common name.
- One dimensional arrays

Syntax: type arrayname[] = new type[size];

Example: int salary[] = new int[20];

Arrays

Initialization of arrays

int number[] ={35,23,67,45,32}

- For initialization of arrays, there is a special value called null which represents an array with no value.
- In Java all arrays store the allocated size in a variable called length.

Example

int l = a.length, where a is an array.

Arrays

Two dimensional arrays

Syntax

Type arrayname[][]=new type[size][size]

Example

int m[][] = new int[4][4]

- ✓ In Java the positions in an array will be automatically set to 0 if it is not assigned a value.

String Handling

- String is a sequence of characters.
- String is the object representation of an unchangeable character array.
- StringBuffer is a companion class used to create strings that can be manipulated after they are created.
- Both String and StringBuffer classes are not subclassible.

String Handling - Constructors

- Empty string creation

```
String s =new String();
```

- String initialized with characters

```
String s =new String("Java");
```

- Strings can be created by passing an array of characters to the constructor.

```
char c[]={'a', 'b', 'c', 'd', 'e', 'f', 'g'};
String s =new String(c);
System.out.println(s);
String s1=new String(c,2,3);
System.out.println(s1);
```

String Handling - Length

```
String s =“abc”;
System.out.println(s.length());
System.out.println(“abc”.length());
```

String Handling - Concatenation

```
String s =“He is ” + age + “years old”;
String s=new StringBuffer(“He is “)
                    .append(age)
                    .append(“ years old.”)
                     .toString()
String s=“four: “ + 2+ 2
String s=“four:”+ (2+2)
```

String Handling – Character Extraction

- To extract single character from a string, we can use the charAt() method.
- To extract more than one character, we can use the getChars() method which allows us to specify the index of the first and one past the last character we want to copy.
- Example Program

String Handling - Comparison

- To check whether two strings are equal, we can use the equals() method. Returns true if two strings are exactly equal.
- If we want to compare strings irrespective of the case, we can use the equalsIgnoreCase() method.
- Example Program

String Handling - Ordering

- For sorting applications, we need to know which string is less than, equal to or greater than the next.
- For this the method compareTo() is used. If the ordering is to be done irrespective of the case, the method compareToIgnoreCase() is used.
- Example Program

String Handling – indexOf() and lastIndexOf()

- Each of these methods returns the index of the character we are searching for. If the search is unsuccessful, it returns -1.
- int indexOf(int ch)- Returns the index of the first occurence of the character ch.
- int lastIndexOf(int ch)- Returns the index of the last occurence of the character ch.
- int indexOf(String str)- Returns the index of the first character of the first occurence of the substring str.
- int lastIndexOf(String str)- Returns the index of the first character of the last occurence of the substring str.

- Example Program

String Handling – indexOf() and lastIndexOf()

- int indexOf(int ch, int fromIndex)- Returns the index of the first occurence after fromIndex of the character ch.
- int lastIndexOf(int ch, int fromIndex)- Returns the index of the first occurence before fromIndex of the character ch.
- int indexOf(string str, int fromIndex)- Returns the index of the first character of the first occurence after fromIndex of the substring str.
- int lastIndexOf(string str, int fromIndex)- Returns the index of the first character of the first occurence before fromIndex of the substring str.

- Example Program

String Copy Modifications

- substring() – used to extract a range from a String using substring.

Example Program

- concat() – creates a new object with the current contents of our string with the new string.

Example Program

String Copy Modifications

- replace() – takes 2 characters as parameters. All occurrences of the first are replaced with the second.

Example Program

- toLowerCase() and toUpperCase() – toLowerCase() converts all the characters in a string to lowercase and toUpperCase() converts all characters to uppercase in a string.

Example Program

- trim() - returns a copy of the string without any leading and trailing whitespace.

Example Program

StringBuffer

- String represents fixed length, immutable character sequences.
- StringBuffer represents growable and writeable character sequences.
- StringBuffer can have characters and substrings inserted in the middle or appended to the end.

StringBuffer Methods

- Constructors – A StringBuffer may be constructed with no parameter which reserves room for 16 chars .
- Can also pass an initial string which will set the initial contents and reserve room for 16 more characters.
- Current length of the StringBuffer can be found using length() method and total allocated capacity using the capacity() method.
- Example Program

StringBuffer Methods

- charAt() and setCharAt() – charAt() method returns the character at a specified position. setCharAt() replace a character at a specified position.
- Example Program

StringBuffer Methods

- append() – used to add substrings to the end of the string.
- Example Program
- insert() – used to insert substrings in a given string.
- Example Program

Packages

- Containers for classes used to keep the class name space compartmentalized.
- Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.

1. Java API packages
2. User defined packages

Java API Packages

- Provides a large number of classes grouped into different packages.

1. java.lang – include classes for primitive types, mathematical functions, threads, exceptions etc.
2. java.util – consists of language utility classes such as date, time, hashtables etc.
3. java.io – provide the facilities for input and output.

Java API Packages

4. java.net – include classes for communication with local computers as well as with internet servers.
5. java.applet – classes for creating and implementing applets.
6. java.awt – include classes for windows, buttons, lists, menus etc.

User defined Packages

1. Declare the package at the beginning of the program using the form

   package pkg1[.pkg2[.pkg3]]; For example, the package java.awt.image should be stored in java\awt\image
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the file as classname.java in the subdirectory created.
5. Compile the file. This creates the .class file in the subdirectory.

Packages – import statement

- Java packages can be accessed using the import statement.
- Syntax

```
import package1[.package2][.package3].classname;
Import packagename.*
```

Packages – Access Protection

	Private	No modifier	Private protected	Protected	Public
Same Class	Yes	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes	Yes
Same package non-subclass	No	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes	Yes
Different package non-subclass	No	No	No	no	Yes

Packages – Access Protection

- Anything declared public can be accessed from anywhere.
- Anything declared private cannot be seen outside a class.
- No modifier is the default.
- Java's protected is equivalent to friend in C++.
- If we want to emulate protected in C++, we must use java's private protected.

Exception Handling

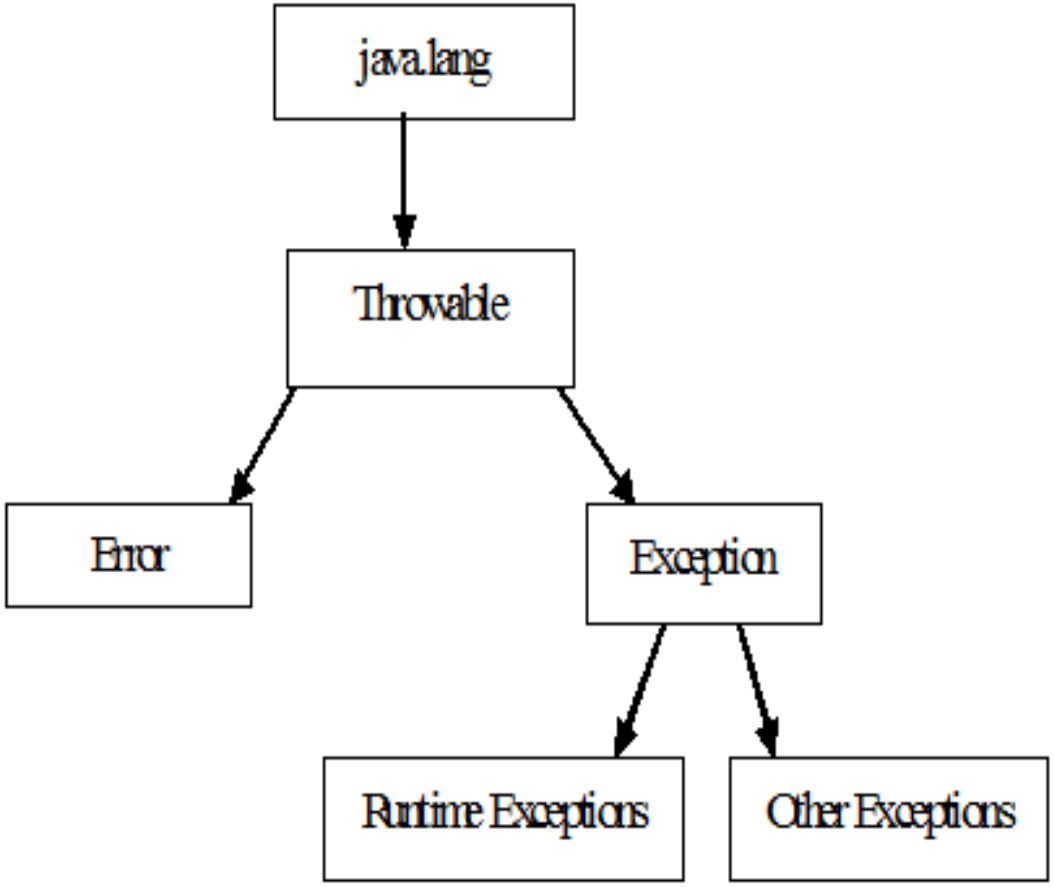
- Exception is an abnormal condition caused by a runtime error in the program.
- Example division by zero.
- When java interpreter encounters such an error, it creates an exception object and throws it. (informs an error has occurred)
- If exception object is not caught and handled properly, the interpreter will display an error message.

Exception Handling

- If we want the program to continue, then we should catch the exception object and display appropriate message for taking corrective actions.

1. Find the problem (Hit the Exception)
2. Inform that an error has occurred (Throws the exception)
3. Receive the error information(Catch the exception)
4. Take corrective actions (Handle the Exception)

Exception Handling - Hierarchy



Exception Types

1. ArithmeticException – caused by mathematical errors such as division by zero.
2. ArrayIndexOutOfBoundsException – caused by wrong array indices.
3. ArrayStoreException – caused when a program tries to store the wrong type of data in an array.

Exception Types

4. FileNotFoundException – caused by an attempt to access a non-existent file.
5. IOException – caused by general I/O failures such as inability to read from a file.
6. StringIndexOutOfBoundsException – caused when a program attempts to access a non existent character position in a string.

Uncaught Exceptions

```
class Exco{
public static void main(String args[])
{int d=0;
int a = 42/d;
}}
```

- An Exception occurs and in the above example, we haven't coded an event handler. Hence default runtime handler is run.

Uncaught Exceptions

Output

Java.lang.ArithmeticException:/by zero at
Exco.main (Exco.java:4)

- Classname Exco, the method name main, the filename Exco.java and line number 4 are all included in the stack trace.

Exception Handling – try and catch

```
try{ statement;
}
catch(Exception e)
{statement;
}//processes the exception
```

- Try is to prepare a block of code that is likely to cause an error condition and throw an exception.

Exception Handling – try and catch

- Catch block catches the exception thrown by the try block.
- Try block can have one or more statements that could generate an exception.
- If any one statement generates an exception, the remaining statements are skipped and execution jumps to the catch block that is placed next to the try block.

Exception Handling – try and catch

- Catch block can have one or more statements.
- If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed.
- Every try statement should be followed by at least one catch statement.

Example program

Exception Handling – multiple catch clauses

- It is possible to have more than one catch statement in try block.

```
try {
statement}
catch(Exception_type1 e)
{statement;//processes exception type 1
}
catch(Exception_type2 e)
{statement; //processes exception type 2
}
catch(Exception_type3 e)
{statement;  // processes exception type 3
}
```

Example program

Exception Handling – Nested try statements

- We can wrap a try statement inside another try statement.
- Each time a try statement is encountered, the context of that exception is stacked up until all of the nested try statement completes.
- If a lower level try doesn't have a match, the stack is unwound and the next try's match is checked.

Example program

Exception Handling- finally

- When exceptions are thrown, the flow of code follows a non linear path.
- finally block is executed always no matter the exception is caused or caught.
- Example Program

Exception Handling – throw and throws

- Throw is used to throw the exception, whereas throws is declarative for the method.
- They are not interchangeable.

Example program

```
public void myMethod(int param) throws MyException{
If (param<10)
throw new MyException("Too Low")
}
```

Exception Handling – throw and throws

- If a method is throwing an exception, it should either be surrounded by a try catch block or the method should have throws clause in its signature.
- Throws clause tells the compiler that this particular exception would be handled by the calling method.

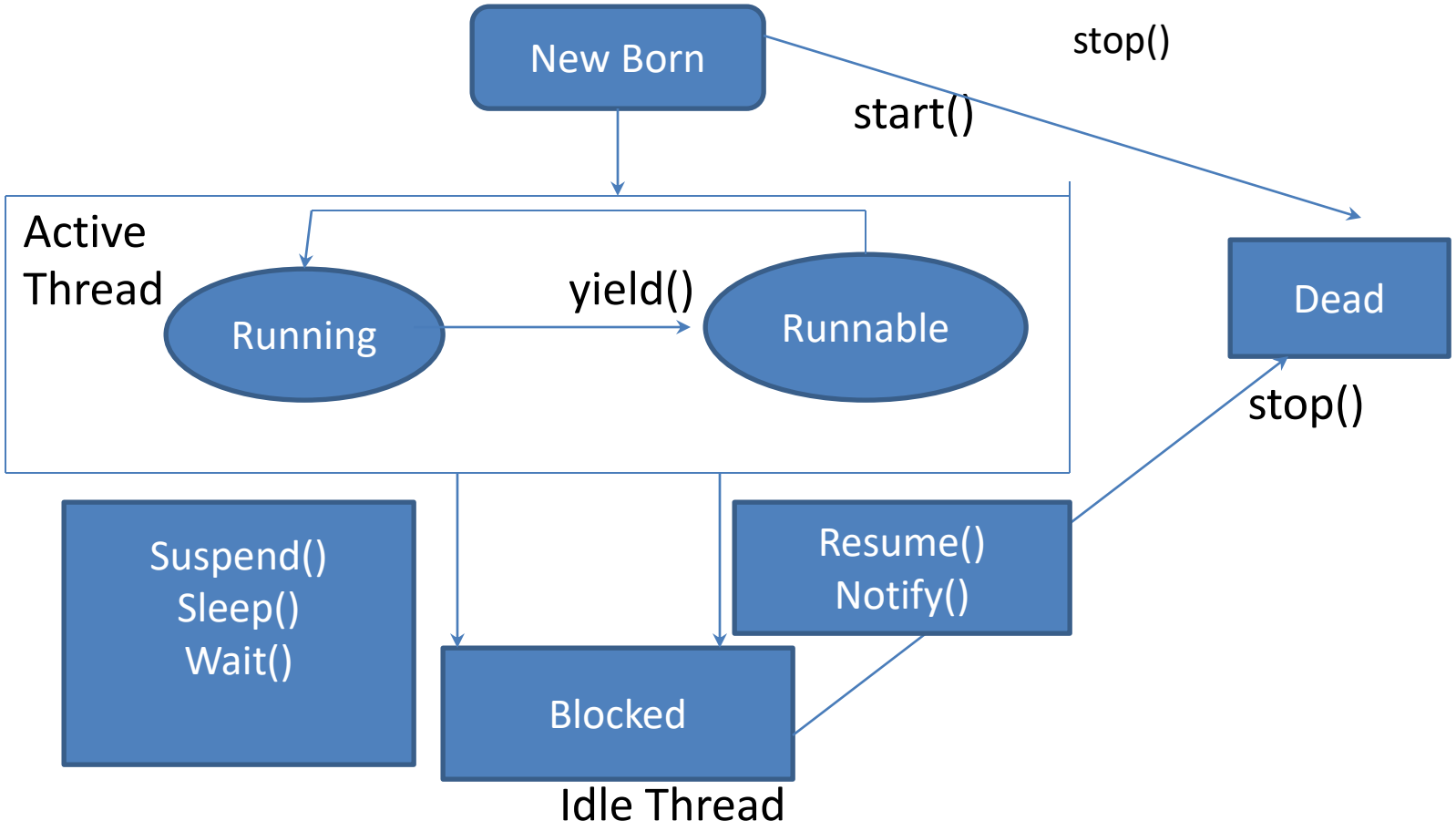
Multithreading

- A concept in which the programs are divided into two or more processes which can be run in parallel.
- For example, when the program waits for the input, the CPU sits idle until the I/P is received.
- In a multithreaded environment, CPU can perform other computational tasks when it waits for the I/O.

Multithreading

- In multitasking, tasks are known as heavy weight processes.
- In multithreading, tasks are known as light weight processes.
- Difference is that heavy weight processes are in separate address spaces and can be thought of as different programs running on the same system like word and excel.
- Threads share the same address space.
- Threads are used in Java enabled Web browsers which can download a file, display a Web page in a window, print another Web page to a printer and so on.

Life Cycle of a Thread



Life Cycle of a Thread

1. **Newborn State**
2. **Runnable State**
3. **Running State**
4. **Blocked State**
5. **Dead State**

Life Cycle of a Thread

- Newborn State – When we create a thread, it is in the new born state.

1. Schedule it for running using start() method.
2. Kill it using stop() method

Life Cycle of a Thread

- Runnable State – This state means the thread is ready for execution and is waiting for the availability of the processor.
- These threads are executed in a FIFO manner.
- If all threads have equal priority, then they are given time slots for execution.
- This process of assigning time to threads is known as time slicing.
- We can relinquish the control to another thread of equal priority by using the yield() method.

Life Cycle of a Thread

- Running State – the processor has given its time to the thread for execution.
- A running thread may relinquish its control in one of the following situations.

1. Suspend using the suspend() method. A suspended thread can be revived using the resume() method.
2. Sleep a thread for a specified time period using the method sleep(time) where time is in milliseconds. The thread re-enters the runnable state as soon as the time period is elapsed.
3. A thread can be put to wait state using wait() method. The thread can be scheduled to run again using notify() method.

Life Cycle of a Thread

- Blocked State – When a thread is prevented from entering into the runnable state and subsequently the running state.
- Happens when the thread is suspended, sleep or in wait state.

Life Cycle of a Thread

- Dead State – Kill the thread using the stop() method.

Creating Threads

- Threads are implemented in the form of objects and contain a method called run().
- The heart of the thread is the run() method.
- Threads can be implemented in 2 ways.

1. By extending the Thread class.
2. By implementing Runnable interface.

Extending the Thread class

1. Declare the class as extending the Thread class.
2. Implement the run() method that is responsible for the execution of the thread code.
3. Create a Thread object and call the start() method.

Example Program

Program to illustrate yield(), sleep() and stop() methods.

Implementing the Runnable Interface

Example Program

End of Unit 3