UNIT III

INTERFACES: MULTIPLE INHERITANCE

Introduction:

- Class in Java cannot have more than one superclass. Java provide alternate approach known as interfaces to support the concept of multiple inheritance.

Defining Interfaces:

- An interface is a kind of class.
- Interface define only abstract methods and final fields.
- Interface do not specify any code to implement these methods and data fields contains only constants.

```
interface interface_name
{
        variables declaration;
        methods declaration;
}
```

- Interface: Keyword
- Interface name: any valid Java variable.
- Variables are declared as

```
static final type variablename=value;
```

- Method declaration will contain only a list of methods without any body statements.

```
return-type methodname1 (parameter list);
```

- Example:

```
interface Item
{
        Static final int code=1001;
        Static final String name= “Fan”;
        Void display();
}
```

Extending Interfaces

- An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
- Syntax:

```
interface name2 extends name1
{
        body of name2
}
```

- Example:

```
interface ItemConstants
{
        int code=1001;
        String name =”Fan”;
}
interface Item extends ItemConstants
{
```

```
        void display();
    }
```

- Also combine several interfaces together into a single interface.
- Example:

```
interface ItemConstants
{
    int code=1001;
    String name =”Fan”;
}
Interface ItemMethods
{
    Void display();
}
Interface Item extends ItemConstants, ItemMethods
{
    -----
    ------
}
```

Implementing Interfaces

- A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.
- Syntax: Class classname implements interfacename

```
{
    Body of the classname
}
```

- Syntax: Class classname extends superclass implements interface1, interface2,…..

```
{
    Body of classname
}
```

Diagram Refer classwork note

Accessing variables in interface

- An interface is a container of abstract methods and static final variables. The interface contains the static final variables. The variables defined in an interface can not be modified by the class that implements the interface, but it may use as it defined in the interface.
- The variable in an interface is public, static, and final by default.
- If any variable in an interface is defined without public, static, and final keywords then, the compiler automatically adds the same.
- No access modifier is allowed except the public for interface variables.
- Every variable of an interface must be initialized in the interface itself.
- The class that implements an interface can not modify the interface variable, but it may use as it defined in the interface

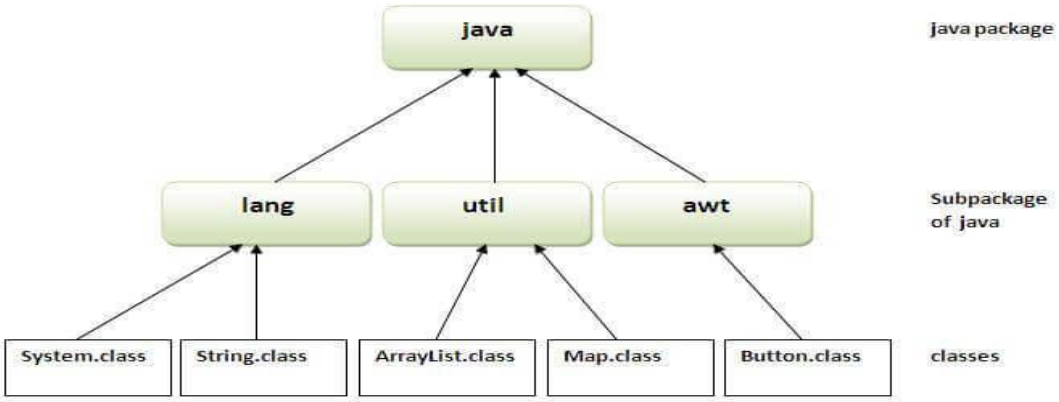
Program: Refer Lab Exercise

PACKAGES

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.



Types Of Packages

- There are basically only 2 types of java packages.
    - System Packages or Java API ′
    - User Defined Packages.

System Packages Or JAVA API

- As there are built in methods , java also provides inbuilt packages which contain lots of classes & interfaces.
- These classes inside the packages are already defined & we can use them by importing relevant package in our program.
- Java has an extensive library of packages, a programmer need not think about logic for doing any task.
- For everything, there are the methods available in java and that method can be used by the programmer without developing the logic on his own.
- This makes the programming easy.

Java System Packages & Their Classes

- **java.lang** : Language Support classes. These are classes that java compiler itself uses & therefore they are automatically imported. They include classes for primitive types, strings, maths function, threads &exception.
- **java .util** : Language Utility classes such as vector, hash tables ,random numbers, date etc.
- **java.io**: Input /Output support classes. They provide facilities for the input & output of data
- **java.awt**: Set of classes for implementing graphical user interface. They include classes for windows, buttons, list, menus & so on.
- **java.net**: Classes for networking. They include classes for communicating with local computers as well as with internet servers.
- **java.applet**: Classes for creating & implementing applets.

User Defined Packages :

- The users of the Java language can also create their own packages.
- They are called user-defined packages.
- User defined packages can also be imported into other classes & used exactly in the same way as the Built in packages

. **i) Creating User Defined Packages**

Syntax :
```
package packageName;
public class className
{ - - - - - - - - - - - - - //
Body of className
- - - - - - - - - - - - }
```

- We must first declare the name of the package using the package keyword followed by the package name.
- This must be the first statement in a Java source file. Then define a classes as normally as define a class.

Example :
```
package myPackage;
public class class1
{ - - - - - - - - - - - - -
// Body of class1
 }
```

- In the above example, myPackage is the name of the package.
- The class class1 is now considered as a part of this package.
- This listing would be saved as a file called class1.java & located in a directory named mypackage.

- When the source file is compiled, java will create a .class file & store it in the same directory.
- The .class files must be located in a directory that has the same name as the package & this directory should be a subdirectory of the directory where classes that will import the package are located.

Steps For Creating Package :

- To create a user defined package the following steps should be involved :-
    - Declare the package at the beginning of a file using the syntax :- package packageName;
    - Define the class that is to be put in the package & declare it public.
    - Create a subdirectory under the directory where the main source files are stored.
    - Store the listing as the classname.java file in the subdirectory created.
    - Compile the file. This create .class file in the subdirectory.

Accessing A Package

- Java package can be accessed either using a fully qualified class name or using a shortcut approach through the import statement.

  Syntax : import package1[.package2][.package3].classname;
- Here, package1 is the name of the top level package, package2 is the name of the package that is inside the package & so on.
- We can have any number of packages in a package hierarchy.
- Finally the explicit classname is specified.
- The import statement must end with a semicolon (;).
- The import startment should appear before any class definitions in a source file.
- Multiple import statements are allowed.

Ex : import firstpackage.secondPackage.Myclass; or import firstpackage.*;

Example: Refer Lab Program

Multithreading

Multitasking

- Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:
    - Process-based Multitasking (Multiprocessing)
    - Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

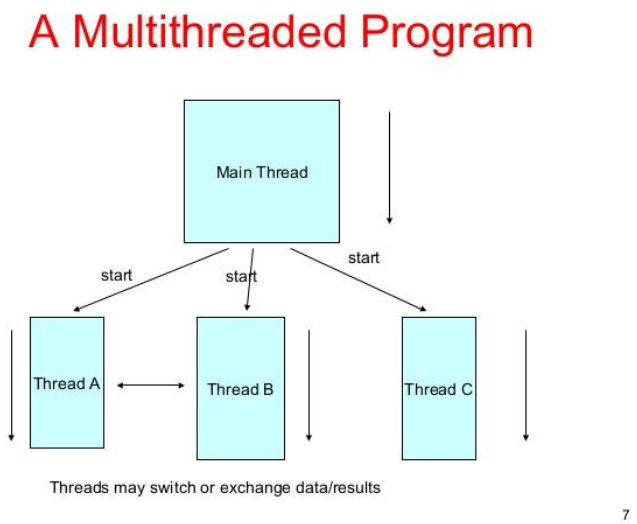
Multithreading

- Multithreading in Java is a process of executing multiple threads simultaneously.
- In reality the processor can do only one thing at a time.
- The processor switches between processes so fast.
- So far discussed programs are in sequential flow .A program begins .sequence of execution and finally ends.
- A thread is a similar program that has a single flow of control.
- It has beginning, a body and an end.
- All the main programs so far we discussed are single threaded program.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared memory area.
- They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process

- Java enable us to use multiple flows of control in developing programs.
- There can be multiple processes inside the OS, and one process can have multiple threads.
- At a time one thread is executed only.
- The threads running in parallel does not really mean that they actually run at the same time.
- All the threads run on single processor , the flow of execution is shared between threads.
- The java interpreter will handle the switching of control between the threads and makes them to run concurrently.
- It is a powerful programming tool.
- It enables the programmers to do multiple things at a time.
- The long program can be divided in to threads and executed in parallel.

Advantages of Java Multithreading

- It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
- You can perform many operations together, so it saves time.
- Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.
- Java Multithreading is mostly used in games, animation, etc.



Creating Threads

- Threads are implemented in the form of object that contain a method called run().
- Java threads can be created in two ways.
    - ***Thread creation using Thread class.***

        - Define a class that extends Thread class and override the run() method with the code required by the thread.
    - ***Thread creation by using Runnable interface***.
        - Define a class that implements runnable interface .The interface has only one method run() with the code to be defined in the thread.
- The approach to be used depends on what class we are creating requires.

Thread creation by extending Thread class

- The Java thread can be created by extending the java.lang.Thread class.
- This gives access to all the thread methods directly.
- The step includes
    - Declare a class an extension of Thread class.
    - Implement the run() method which is responsible for executing the code that the thread will execute.
    - Create the thread object and call the start()method to initiate the thread execution.

i) Defining the class

- The Thread class can be extended as follows
- Syntax: class userthread extends Thread
  { …… }
- Example: class Mythread extends Thread
  { …….. }
- The above declaration will create a thread Mythread which is the extension of the Thread class.

ii) Implementing the run() method

The run method has been inherited by the class Mythread.

We have to over ride the method in order to implement the code to be executed by the thread.

The implementation is as follows

```
public void run()
{
…….  //thread code here
}
```

When we start a new thread Java calls the thread's run() method.

The run() method is the heart and soul of the thread.

It makes the entire body of the thread.

It is the only method in which the thread's behaviour can be implemented.

iii) Starting a new thread

- To actually create and run an instance of our class Thread we must write the following code.

```
Mythread m = new Mythread();
m.start();
```

- The first line instantiates a new object of class Mythread.
- This statement creates the object.
- The thread will this object. The thread is a new born state.
- The second line calls the start() method. This cause the thread to move to the runnable state.
- The Java runtime will schedule the thread to run by invoking its run() method.
- Now the thread is in running state.

Example: Refer Lab Program

Stopping and Blocking Thread

Stopping a Thread

- Whenever we want to stop a thread from running we just call the method stop().

```
Ex: threadobject.stop();
```

- This object can cause the thread to move to the dead state.
- The thread may also go to the dead state when it reaches the end of its method.
- The stop() method may also used for premature death of a thread.

Blocking a thread

- A thread can be temporarily blocked or suspended from entering in to the runnable and subsequently running state by using either of the following methods
  - sleep() – blocked the thread for a specified time.
  - suspend()- blocked the thread until further orders.
  - wait() - blocked the thread until certain condition occurs.
- The above methods will make the thread to go to blocked (or non runnable) state.
- The thread will return to runnable state when spacified time is elapsed in case of sleep().
- The resume()method is invoked in case of suspend().
- The notify() method is called in case of wait().

Life Cycle of a Thread

- During the life time of a thread, the thread can enter many states.
- The life cycle of the thread in java is controlled by JVM. They include
  - New born state
  - Runnable state
  - Running state
  - Blocked state
  - Dead state

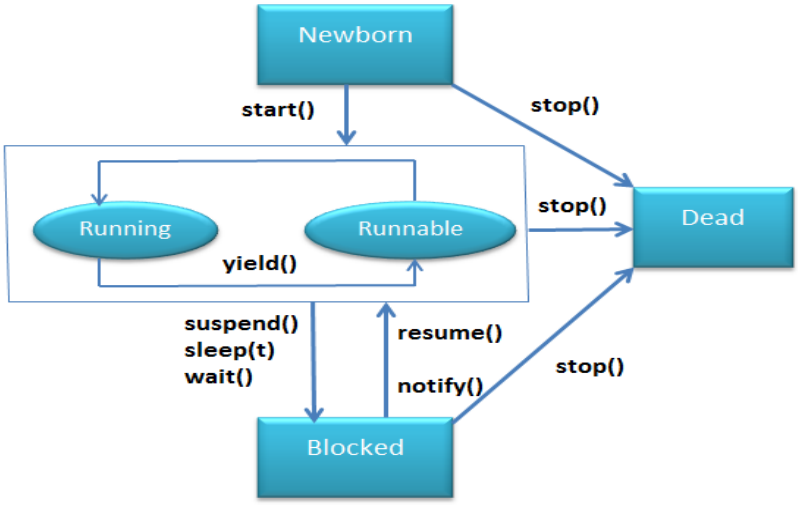
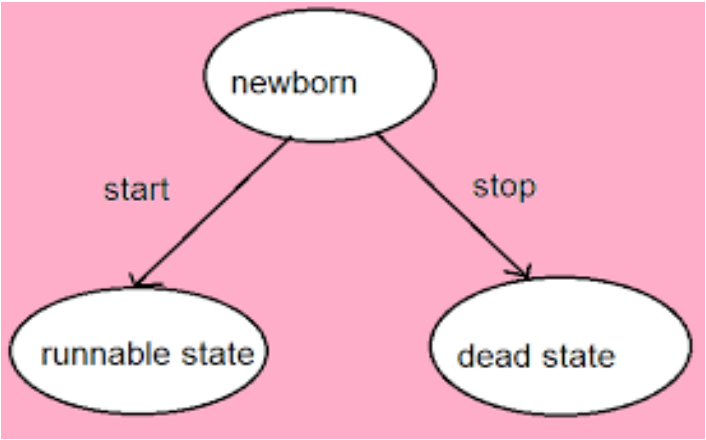


Fig: Life Cycle of Thread

New born state

- When we create a thread object, the thread is born and is said to be in new born state.
- The thread is not yet scheduled for running.
- At this state, we can do any of the following things.
  - Schedule it for running using start() method.
  - Kill the thread by using stop() method.
- If scheduled it moves to runnable state.



Runnable Thread

- The runnable thread means the thread is ready for execution and it is waiting for the availability of the processor.
- The thread has joined the queue of threads that are waiting for execution.
- If all the threads having equal priority then they are given time slots for execution in round robin fashion (ie)FCFS
- That thread that relinquishes its control and joins in queue at the end and again waits for its turn.
- The process of time to threads is called time slicing.
- The thread can automatically relinquish its control to equal priority or higher priority before its turn comes.
- This can be done by yield() method.

Running state

- The thread runs until it relinquishes it control on its own or it is preempted by a higher priority thread.
- A running thread may relinquishes its control in one of the following situations.

1.The thread can be suspended by suspend() method.

A suspended thread can be revived by resume() method.

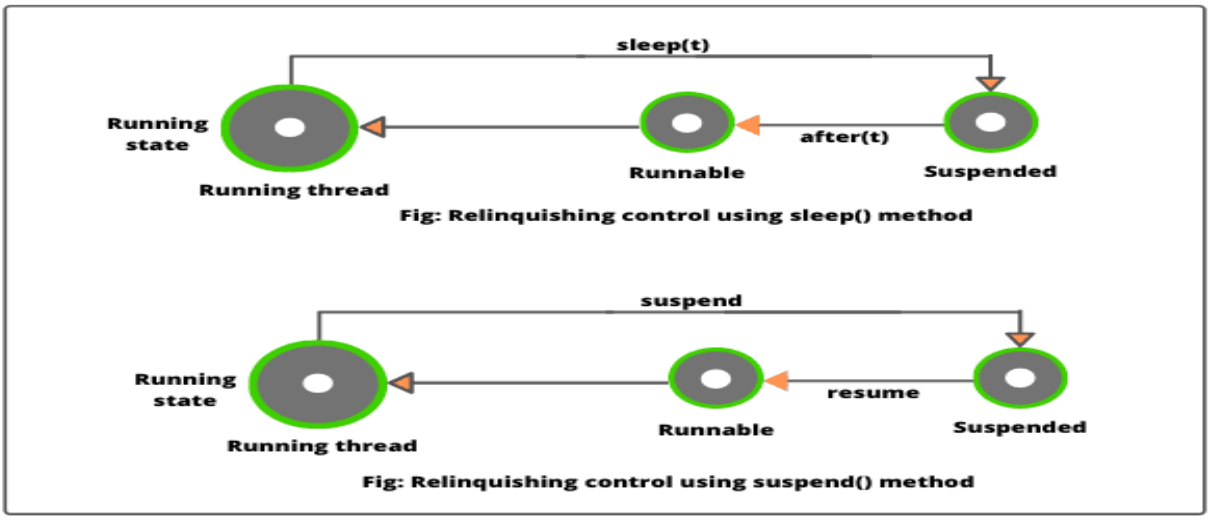
This is used when we want to suspend a thread for some time for certain reason and don't want to kill it.

2.The thread can be made to sleep().

It can be made sleep for a specified time period using sleep(time) method

The time is in milliseconds.

The thread reenters the runnable state as soon as this time period is elapsed.



3.It has been made to wait until some event occurs.

This is done by wait() method.

The thread can be scheduled to run again using notify() method.

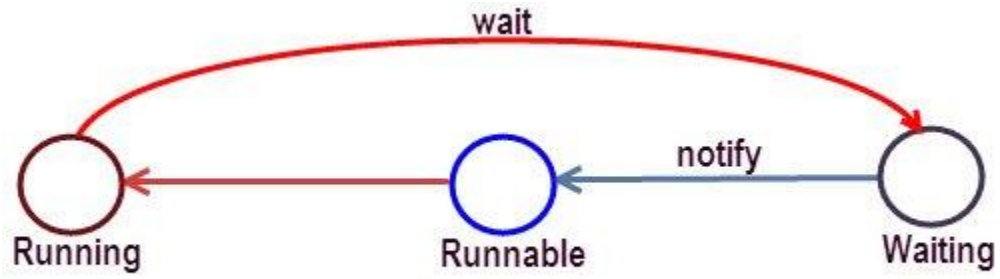


Fig. Control with wait() method

Blocked state

- A thread is said to be blocked when it is prevented from entering in to the runnable state and subsequently the running state.
- This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements.
- A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

Dead state

- Every thread has a life cycle.
- A running thread ends its life when it has completed executing its run() method.
- It is a natural death.
- The thread can be killed by sending stop message to it at any state thus causing a premature death to it.

- A thread can killed as soon as it was born or while it is running or even in the runnable state.

Using Thread Methods

- There are various methods which can be called on Thread class object.
- These methods are very useful when writing a multithreaded application.
- Thread class has following important methods.

String getName()	Retrieves the name of running thread in the current context in String format
void start()	This method will start a new thread of execution by calling run() method of Thread/runnable object.
void run()	This method is the entry point of the thread. Execution of thread starts from this method.
void sleep(int sleeptime)	This method suspend the thread for mentioned time duration in argument (sleeptime in ms)
void yield()	By invoking this method the current thread pause its execution temporarily and allow other threads to execute
void join()	This method used to queue up a thread in execution. Once called on thread, current thread will wait till calling thread completes its execution
boolean isAlive()	This method will check if thread is alive or dead

Thread Priority

- In a Multi threading environment, thread scheduler assigns processor to a thread based on priority of thread.
- Whenever we create a thread in Java, it always has some priority assigned to it.
- Priority can either be given by JVM while creating the thread or it can be given by programmer explicitly.
- Accepted value of priority for a thread is in range of 1 to 10.
- There are 3 static variables defined in Thread class for priority.
- **MIN_PRIORITY**: This is minimum priority that a thread can have. Value for this is 1.
  NORM_PRIORITY: This is default priority of a thread if do not explicitly define it. Value for this is 5.
  MAX_PRIORITY: This is maximum priority of a thread. Value for this is 10.

- public final int getPriority(): java.lang.Thread.getPriority() method returns priority of given thread.
- public final void setPriority(int newPriority): java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority.

```
class A extends Thread
{
        public void run()
        {
                for(int i=1; i<=5 i++)
                {
                        if(i=1) yield();
                        System.out.println("From Thread A   i="+i);
                }
                System.ou.printnln("Exit from A");
        }
}
class B extends Thread
{
        public void run()
        {
                for(int j=1; j<=5 j++)
                {
                        System.out.println("From Thread B   j="+j);
                        if(j=3) stop();
                }
                System.ou.printnln("Exit from B");
        }
}
class C extends Thread
{
        public void run()
        {
                for(int k=1; k<=5 k++)
                {
                        System.out.println("From Thread A   i="+i);
                        if(k==1)
                        {
                                sleep(1000);
                        }
                }
        System.ou.printnln("Exit from C");
        }
}
```

```
//Thread Priority
class Threaddemo
{
	public static void main(String args[])
	{
		A a = new A();
		B b = new B();
		C c = new C();
		c.setPriority(Thread.MAX_PRIORITY);
		b.setPriority(a.getPriority()+1);
		a.setPriority(Thread.MIN_PRIORITY);
		System.out.println("Start thread A");
		a.start();
		System.out.println("Start thread B");
		b.start();
		System.out.println("Start thread C");
		c.start();
	}
}
```

Implementing the Runnable Interface:

- The runnable interface declare the run() method that is required for implementing threadsin our programs.
- The steps include:
  - Declare the class as implementing the **Runnable** interface.
  - Implement the **run()** method.
  - Create a thread by defining an object that is instantiated from this "runnable" class as the target of the thread
  - Call the thread's **start()** method to run the thread.

Example: Refer Class Work Note