*MTSM25 - Notepad

File Edit Format View Help

page 1:
****

NOTES:
======

MULTITHREADING:

By extending thread classs:(thread class is implementing runnable interface)
****
1) Main thread start method used to create child thread .(Main thread == main method)
2) to execute child logic ,start method which is in parent class(i.e.Thread class)will call
        run method of child class. .
        note-Thread class also has his run method which has empty implementation,we are overriding
        run method in child class to execute our logic

Q.what if we call t.run method directly insted of t.start()?
--> then that run method will act as normal method ,and here we are not able to create any
thread .(we need to call start method for child class which is extending Thread,then only new thread
is created)

3)note: Thread class start method will call no argument run method only.
if we want to call argument run method the we need to call it explicitly(how to call)

4) If we override start method in child class then no thread is created coz child start method
will not have logic to create thread.it will act as normal method.

note: It not recomended to overide start method,othervise dont go for multithreading concept

5)if we try to retart same thread by using start method again then we will get -->IllegalThreadState exception i.e runtime

By Implementing runnable interface:(directly we are implementing runnable)
****
1) runnable interface present in java.lang package and it contains only one methode i.e run method
2)

```
class MyRunnable implements Runnable{
        public void run(){
                for(int i = 0;i<10;i++){ -------|
                        SOUT("child");          \ job executed by child thread
                        }                       /
                }----------------------------|

        }
```

page 2:
****

```
class ThreadDemo{
        public static void main (Strinng[args]){
```

*MTSK25 - Notepad

File Edit Format View Help

page 2:
****

```
class ThreadDemo{
        public static void main (Strinng[args]){
                myRunnable r = new myRunnable();
                Thread t = new Thread(r);     // r is target runnable  Q.how thread class allowing to run run() method of child here
                t.start();

                for(int i = 0;i<10;i++){
                        SOUT("main thread");
                        }
                }
}
```

->above if you not ppass object r as parameter ,then start() method will call empty run method of start class.
->as we are passing r as parameter then it is performing run method of Myrunnable class.

3) we will get mixed output & we cant tell exact output.

4) case study:

```
MyRunnable r = new MyRunnable();
Thread t1 = new Thread();
Thread t2 = new Thread(r);
```

case 1: t1.start();
 -> new thread created and which is responsible for execution of thread class run method,
 whiich has empty implementation

case 2: t1.run();
-> no new thread will be created and thread class run method will be executed just like
a normal method call.

case 3: t2.start();
->A new thread will be created which is responsible for the exceution of MyRunnable class run method.

case 4: t2.run();
-> A new thread wont be created and MyRunnable run method will be eexecuted just like a normal method call.

case 5 : r.start()
-> will we get compile time error saying ,my runnable class does'nt have start capability.
CR. cannot find symbol : method start () location class MyRunnable

case 6: r.run();

page3:
******

"M15M25 - Notepad

File Edit Format View Help

page3:
****

-> no new thread will be created,and my runnable run method will be executed likke normal method call.

5) which appproch is best to define a thread?
--> among two ways of defining a thread, implements runnable appproch is recomended.
-In the first approch our class always extends thread class,there is no chance of extending in any other class.hence we are missing inheritance benifit.
-but in the second approch while implementing runnable interface we can extend any other class .hence we wont miss any Inheritance benifit.
-Beacause of above reason Impplementing runnable Interface Approch is rcomended than extending thread class.

Thread Class Constructors:
==========

1)Thread t = new Thread();
2)----------/----------(Runnable r);
3)----------/----------(String name);
4)----------/----------(Runnable r,String name);
5)----------/----------(ThreadGroup g,String name);
6)----------/----------(ThreadGroup g,Runnable r);
7)----------/----------(ThreadGroup g,Runnable r,String name);
8)----------/----------(ThreadGroup g,Runnable r,String name,long stackSize);

*Durgas Approch to define a thread(not recomended to use)

```
class MyThread extends Thread {
        public void run(){
                sout("child thread");
                }
        }

class ThreadDemo{
        public static void main(String args[]){
                MyThread k = new MyThread();
                Thread t = new Thread(k);     // r is target runnable  Q.how thread class allowing to run run() method of child here
                t.start();

                for(int i = 0;i<10;i++){
                SOUT("main thread");
                }
        }
```

It also valid ,just lie runnable but we are extending child thread class with Thread class

ppage 4:
****
Getting and Setting name of a thread:

*M15M25 - Notepad

File Edit Format View Help

ppage 4:
****

Getting and Setting name of a thread:
---

1) every thread in java has a some name,it may be default name generated by JVM or customised name provided by programer.
2) we can get and set name of thread by using the following two methods of thread class
    a) public final string getName()
    b) public final void setName(String name)

```
class MyThread extends Thread{
}
class Test{
    public static void main(String args[]){
    System.out.println(Thread.currentThread().getName());
    MyThraed t = new MyThread();
    Sysout(t.getName());
    thread.currentThread().setName("pawan kalyan");
    Sysout(Thread.currentThread().getName());
}
```

-> we can get current executing thread object by using thread.currentThread() method

```
Class MyThread extends Thread{
    public void run(){
    System.out.println("run method executed by thread:" + Thread.currentThread().getName());
}
}

class Test{
    public static void main(String args[]){
    Mythread t = new MyThread();
    t.start();
    Sysout("main thread executed by thread" + Thread.currentthread().getName());
}
}
```

o/p : main method.....: main
      run method .....:Thread-0

Synchronization
==========

-> Synchronized is a modifier applicable only for methods and blocks,but nor for calsses and variable.
-> if a multipple threads are trying to operate simultaniously on th same ava obect then their may be a chance of data inconsistency problem.
-> To overcome this problem we should go for syncronised keyword.

ppage 5:
****

*M15M25 - Notepad

File Edit Format View Help

ppage 5:
****

-> if a method or block declared as syncronized then at a time only one thread is allowed to execute,
that method or block on the given object,so that data inconsistancy problem will be resolved.
-> The main advanatge of syncronised keyword is we can resolve data inconsistency problems,but
the main disadvantage of syncronised keyword is it increases waiting time of threads and creates
performance problems.Hence If ther is no specific requirnments then it is not recomended to use
syncronised keyword.

-> Internally syncronization concept is implemented by using lock.
Every object in ava has unique loc
-> Whenever we are using syncronised keyword,then only locked concept will come into the picture.
-> if a thread wants to execute syncronized method on the given obect first it has to get lock of that objject.
once thread got the lock then it is allowed to excuute any syncronized method on that obect.
-> Once method execution completes ,automatically thread releases a lock.
-> aquviring and releasing lock internally takes care by jvm.and pprogramer not responsible for this activity.

Comparator:
==========
1) comparator present java.util package and it define two methods compare and equals
2) public int compare(Obect obj,Obect obj)
        return -ve iff obj has to come before obj2
        return +ve if obj has to come after obj2
        return 0 if both are same
3) wheneever we are implementing comparator interface compulsory we should implementation only for compare method
and are not required to provide implementation for equals method becoz it is already available to our classs
from object class through inheritence.

Covarient return type:
=====================

1)till 1.4 version java allowed same return type only for overriding.
2) from 1.5 version covarient return type allowed.According to this child class method return type need not same as
parent class method return type.Its(parent) child typpe also allowed

Note : we can compile code in whichever version of java
javac -source 1.4 p.java

3)ex class A{
        public Obect m1(){
                return null;
                }
        }

page 6:
****

*M1SM25 - Notepad
File Edit Format View Help

page 6:
****

```
        class B extends A{
        public String m2(){
                return null;
                }
        }
```

this above example only invalid til 1.4 version.

4)

	parent class return type	child class return type	valid or not
1)	Object	Object/String/StringBuffer	valid
2)	String	Object	no valid(String is not parent of Obect class)
3)	Number	Number/Integer	Valid
4)	double	int	both are data types not classes

5) Covarient return type concept applicable only for object type but not for premitive types.
6) Based our requirment ,we can define exactly same private method in child class ,it is valid but not overriding.(as both private method has their access inside there class only)

Inner Classes:
==========

1) sometime we can declare a class inside another class such type of classes are called inner classes.
2) Inner classes concet introduced in 1.1 version to fix gui bugs as a ppart of event handling ,but bcoz of powerful features,and benefits of inner class slowly programer started using in regular coading also.
3) Without existing one typpe of object if there is no chance of existing another type of object,then we should go for inner classes.
4)

example 1: University consist of several deppartments ,without existing university there is no chance of existing deppartment. hence we have to declare department class inside univercity class.

```
class University{   ----------------outer class
        class department{-----------inner class
                }
        }
```

example 2: Without existing car object their is no chance of existing engine object,hence we have to declare engine class inside car class.

```
class Car{
        class Engine{
                }
        }
```

page 7:

*M15M25 - Notepad

File Edit Format View Help

page 7:
****

exampple 3: Map is group of key value pairs and each key value pair is called an entry.Without existing map obect is no chance of existing entry object.hence inteface entry is define inside map interface.

```
interface Map{
        interface entry{
                }
        }
```

Note: 1) without existing outer class object there is no chance existing inner class object
2) their relation bettwen outer class and inner class is not "IS-A" relation ,it is "HAS-A" relationship(composition or aggrigation).

5) Based on position of declaration and behaviour all inner classes devide into four type
1) normal or regular inner classes
2) method local inner classes.
3) anonymous Inner classes
4) Static nested classes

a)Normal or regular inner classes;
==========

-> if we are declaring any named class directly inside a class without static modifier,such type of inner class is called normal or regular inner classs.

example 1:
==========

```
class Outer{
        class Inner{

        }
}
```

=> after performing compilation

javac Outer.java -- > we get two .class file
1) Outer.class
2) Outer$Inner.class

=> if we perform run

java outer --> RE: NoSuchMethodError:main
java outer$Inner --> RE: NoSuchMethodError:main

example 2:
==========

page 8:
****

*M15M25 - Notepad
File Edit Format View Help

page 8:
****

```
class Outer{
        class Inner{

        }
        public static void main--...{
                sout("Outer class main method");
        }
}
```

=> after performing compilation

javac Outer.java -- > we get two .class file
  1) Outer.class
  2) Outer$Inner.class

=> if we perform run

java outer --> Outer class main method
java outer$Inner --> RE: NoSuchMethodError:main

example 3:
=========

```
class Outer{
        class Inner{                                    <-- imagin inner class as instance compponent
                public static void main--...{
                        sout("Inner class main method");
                        }
                }
        }
```

=> after performing compilation

javac Outer.java -- > CE: Inner classes cannot have static declaration

->Inside inner class we can not declare any static members,hence we cant declare main method and we cant run inner class directly from command prompt

Case 1: Accessing inner class code from static area of outer class:
------
page 9
****

*M1SM2S - Notepad
File Edit Format View Help

page 9
****

```
class Outer{
        class Inner{
                public void m1(){
                        sout("Inner class main method");
                        }
                }

                p s v main(String[] args){

                Outer o = new Outer();
                Outer.Inner i = o.new Inner();
                i.m1();
        }
```

Outer.Inner i = new Outer().newInner()

new Outer().new Inner.m1()h

case 2: Accesing inner class code from instace area of outer class
------

```
class Outer{
        class Inner{
                public void m1(){
                        Sout("Inner class method");
                                }
                }
        public void m2(){
                Inner i = new Inner();
                i.m2();
        }
        p s v main(String [] args){
                Outer o = new Outer();
                o.m2();
                }
        }
```

case 3: Accessing Inner class code from outside of outer class :(Similar to first case)
----

page 10:
****

```
class Outer{
        class Inner{
                public void m1(){
```

*M15M25 - Notepad

File Edit Format View Help

page 10:

****

```
class Outer{
        class Inner{
                public void m1(){
                        sout("Inner class method");
                        }
                }
        }

class Test{
        p s v main(String[] args){
                Outer o = new Outer();
                Outer.Inner i = o.new Inner();
                i.m1();
                }
        }
```

-> from normal or regular iinner class we can access both static and non static members of outer class directly.

```
class Outer{
        int x = 10;
        static int y = 20;
        class Iner{
                public void m1(){
                        sout(x);
                        sout(y);
                }
        }
        p s v m(){
                new Oter().new Iner().m1();
        }
}
```

o/p; 10 20

-> within the inner class ,this alsways refer current inner class object.

-> if we want to refer current outer class object ,we have to use outerclassname.this .

example:

```
class Outer{
        int x = 10;
        class Inner{
                int x = 100;
                public void m1(){
                        int x= 1000;
```

File Edit Format View Help

```
        class Inner{
                int x = 100;
page 11;
*********

                public void m1(){
                        int x= 1000;
                        sout(x);            //1000    -->  also we can write -> Inner.this.x
                        sout(this.x);       //100
                        sout(Outer.this.x) //10    <-- Imp observation (super not allowed as inheritance not present)
                        }
                }
        p s m v(){
                new Outer().new Inner.m1();
                }
        }
-> the only applicable modifiers for outer classes are  public,default,final,abstract,strictfp
-> but for inner classes applicable modifiers are public ,default,final,abstract,strictfp,private ,protected ,static

Anonymous Inner class:
=====================
-> the main purpose of anonymous inner class is just for instant use(one time usage)
-> Based on declaration and behavior ,ther are thrtee type of anonymous inner classes.
1) Anonymous Inner class that extends a class
2) Anonymous Inner class that implements an interface
3) Anonymous inner class that defined inside arguments

1) Anonymous inner class that extends a class:
----------------------------------------------

Thread t = new Thread(){

};   <---- it will going to act as child class of thread class.

-----------------------------
Q.what are top level class?
-----------------------------

class Popcorn{
        public void taste(){
                System.out.println("salty");
                }
}

class Test{
        public static v m(){
                Popcorn p = new Popcorn(){
page 12;
*********
```

```
File  Edit  Format  View  Help
            Popcorn p = new Popcorn(){
page 12:
*********

                    p v taste(){
                            sout("spicy");
                            }
                    };
            p.taste();//spicy
            Popcorn p1 = new Popcorn();
            p1.taste(); //salty
            Popcorn p2 = new Popcorrn(){
            p v taste(){
                    sout("sweet");
                     }
                   };
            p2.taste(); //sweet
            Sout(p.getClass().getName()); //Test$1
            Sout(p1.getClass().getName());//Popcorn
            Sout(p2.getClass().getName());//Test$2
        }
}

The generated .class files are popcorn.class,Test.class ,Test$1.class,Test$2.class.

Analysys;
---------
1) Popcorn p = new Popcorn():
-> just we are creating popcorn objecct
2)popcorn p = new popcorn(){
        };
-> we are declaring a class that extends a popcorn without name(anonymous inner class)
->for that child class we are creating an object  with parent reference.

3) Popcorn p = new popcorn(){
        p v taste(){
                sout("spicy");
                }
        };

->we are declaring a class that extends popcorn without name(anonymous inner class)
-> in that child class we are overriding taste method
-> for that child class we are creating an object with parent reference.


page 13
***********

Defining a thread by Extending Thread class:
```

File Edit Format View Help

page 13
****

Defining a thread by Extending Thread class:
---

normal class approch:

```
class myThread extends Thread{
        p v run(){
                for(int i = 0;i<0;i++){
                        sout("child thread");
                        }
                }
        }

class ThreaDemo{
        p s v m(){
                myThread t = new myThread();
                t.start();
                for(int i = 0;i<0;i++){
                        sout("main thread");
                        }
                }
        }
```

anonymous inner class approch:

```
class ThreadDemo{
        p s v main(){
                Thread t = new Thread (){
                        p v run(){
                                for(int i = 0;i<0;i++){
                                        sout("child thread");
                        }
                }
        };
        t.start();
        for(int i = 0;i<0;i++){
                sout("main thread");
                }
        }
}
```

page 14:
****

```
}
page 14:
********

(refer page 2 for normal runnable implementation)

anonymous inner class aproch:
------------------------------

class ThreadDemo{
        p s v main(){
                Runnable r = new Runnable (){
                        p v run(){
                                for(int i = 0;i<0;i++){
                                        sout("child thread");
                        }
                }
        };
        Thread t = new Thread(r)
        t.start();
        for(int i = 0;i<0;i++){
                sout("main thread");
                }
        }
}

anonymous inner class that define inside argument:
-----------------------------------------------
class ThreadDemo{
        p s v main(){
                new Thread(new Runnable (){
                        p v run(){
                                for(int i = 0;i<0;i++){
                                        sout("child thread");
                        }
                }
        }).start();
        for(int i = 0;i<0;i++){
                sout("main thread");
                }
        }
}
```