Unit-3
Generics and Multithreading

Topics to cover

- *| Motivation for generic programming
- *| Generic classes
- *| Generic Methods
- *| Generic code and virtual Machine
- *| Inheritance and generics
- *| Reflection and generics
- *| Multi threaded programming
- *| Interrupting thread
- *| Thread States
- *| Thread properties
- *| Thread Synchronization
- *| Executors
- *| Synchronizers

Generics

*| Generic is about writing codes that is safe, reusable and works with many data types
*| Generic Programming enables the programmer to create classes, interfaces and methods that can automatically work with all types of data
*| The Goal: It has expanded the ability to reuse the code safely and easily.

Advantages of Java Generics

There are mainly three fantastic advantages
1) Type Safety
2) Type Casting is not required
3) Compile-Time checking

Code: Simple Generic class

```
class TwoGen<T,V>
{
  TObj;
  VObj2;
  TwoGen(TO1, VO2){
     ob1 = O1;
     ob2 = O2; }
void showTypes (){
     S.O.P("Type of T : " + ob1.getClass().getName());
     S.O.P("Type of V is : " + ob2.getClass().getName()); }
T getOb1()
{ return ob1; }
V setOb1() {
     return ob2; }
Public class mainclass {
          public static void main (String[] args){
          TwoGen<Integer, String> tgObj = new TwoGen
                                  <Integer, String> (88, "Generics");
          tgObj.showTypes();
          int v = tgObj.getOb1();
          S.O.P("value : " + v);
          String str = tgObj.getOb2();
          S.O.P("value : " + str);
}
```

output:

Type of T : Java.lang.Integer
Type of V : Java.lang.String
Value : 88
value : Generics

Code : Bounded Types :

This is used to restrict the type parameter to avoid errors

```
Class Stats <T extends number>
    T[] nums;
    Stats (T[] n){
        nums=n;
    }
    double average(){
        double sum=0.0;
        for(int i=0;i<nums.length;i++){
            sum+=nums[i].doubleValue();
        }
        return sum/nums.length;
    }
}

Public class MainClass{
    Public Static void main(String[]args){
        Integer inums[]={1,2,3,4,5};
        Stats<Integer>iob=new Stats<Integer>(inums);
        double v=iob.average();
        S.O.p("Iob average is :"+v);
    }
```

output average : 3.0

Generic methods:

*) Methods defined with type parameters right before the return type

Example:

```
                    ↗Type
public static <T> void printArray ( T[] elements ) {
        S.O.P ("Array :");
        for ( T elements : elements) {
                S.O.P (elements + );
        }
        S.O.P ();
}

public static void main (String [] args) {
        Integer iarr = {10, 20, 30, 40, 50};
        printArray ( iarr);
```

output:

Array : 10, 20, 30, 40, 50

Generic code and Method reading

*) The JVM doesn't know about generics at runtime. The compiler replaces all type parameters with their bounds. This process is called type Erasure

*) Effect: This ensures backward compatibility with older Java versions

*) A method can be overloaded based on types

Snippet for Erasure type:

```
public static void main (String [] args) {
        List <String> stringlist = new ArrayList <> ();
        List <Integer> integerlist = new ArrayList <> ();
}
```

Inheritances and Generics:

*) List<String> is not a subtype of List<Object>. To express flexible relationships, we use wildcard (?)

Snipet

```
public static void printList (List<?> list){
        S.O.P (list);
}
```

Reflection and Generics:

```
import java.util.*;
```

Simply we are retrieving type!

Restriction and Limitation:

*) Can not use primitive Types
*) Cannot create arrays of parameterized Types
*) Cannot Instantiate type parameters
*) Cannot declare static fields

* End of Generics :- *

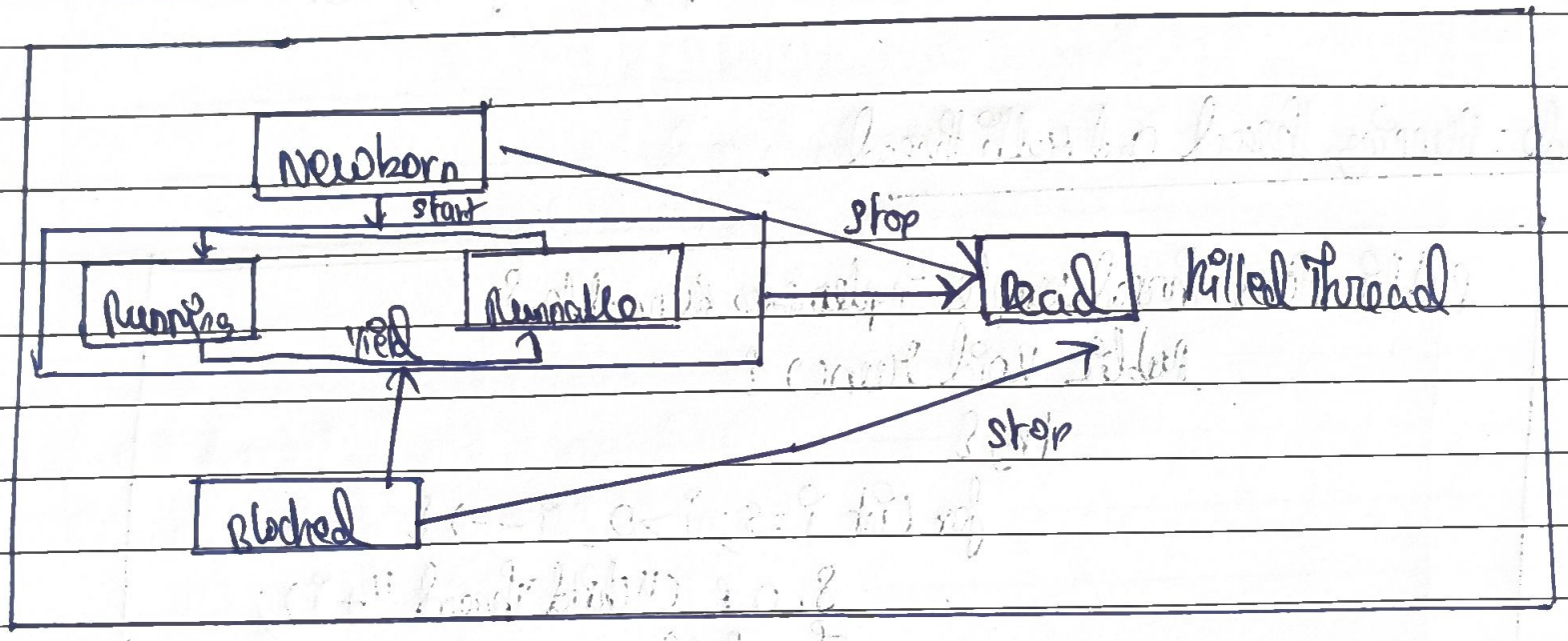
Multi Threaded program

* Multithreaded programming involves dividing a program into two or more subprograms (threads), which can execute concurrently (in parallel)

Core concepts:

1) Thread
2) Multithreading
3) Multitasking

The Java Thread Model



1) newborn state: when a thread object is created, a new thread is born enters the state
2) Runnable state: The Thread is ready for execution and waiting for the processor. If threads have the same priority, they are executed in round robin fashion
3) Running state: The processor has given its time to the thread for execution
4) Blocked State: The thread is prevented from entering the runnable state. This occurs if the thread is suspended, sleeps or waits for an event
5) Dead State: A runnable thread enters this state when it completes its task or otherwise stops

Code: Main thread

```
class MainThread demo {
    public static void Main (String[] args) {
    Thread T1 = Thread.currentThread();
    t1.setName ("MainThread");
    S.O.P ("Name of the Thread is " + t1);
    }
}
```

Output: Name of the Thread is [MainThread, 5, main]
↓ ↓ ↓
name priority class

Code: Running thread and multi thread

```
Public class ThreadSample implements Runnable {
    public void run() {
        try {
            for (int i=5; i>0; i--) {
                S.O.P ("child Thread " + i);
                Thread.sleep(1000);
            }
        } Catch (InterruptedException e) {
            S.O.P ("child interrupted");
        }
        S.O.P ("Exiting child Thread");
    }
}
```

```
public class MainThread {
    public static void main (String[] args) {
        ThreadSample d = new ThreadSample();
        Thread s = new Thread(d);
        s.start
        try {
            for (int i=5; i>0; i--) {
                S.O.P("Main thread "+i);
                Thread.sleep(5000);
            }
        }
        Catch (InterruptedException e) {
                S.O.P("Main interrupted");
        } S.O.P("Exiting Main Thread");
    }
}
```

Output:

child Thread 5
Main Thread 5
child Thread 4
:
child Thread 1
Main Thread 4
... and soon

Thread priority

* Each thread has priority based on that threads will be executed in ords
  Norm priority (medium)
  Min priority: 1
  Max priority: 10 → Executes first

Code: Priority

```
public class MyThread1 extends Thread {
    public void run() {
        for (int i=0; i<5; i++) {
            Thread cur = Thread.currentThread();
            cur.setPriority(Thread.MAX_PRIORITY);
            S.O.P("Thread name : " + Thread.currentThread().
                                    getName());
            S.O.P("Thread Priority : " + cur);
        }
    }
}

class MyThread2 extends Thread {
    //Constructor
    public void run() {
        for (int i=0; i<5; i++) {
            Thread cur = Thread.currentThread();
            cur.setPriority(Thread.MIN_PRIORITY);
            S.O.P(cur.getPriority());
            S.O.P("Thread name : " + Thread.currentThread().
                                    getName());
            S.O.P("Thread Priority " + cur);
        }
    }
}

public class ThreadPriority {
    public static void main(String[] args) {
        MyThread1 m1 = new MyThread1("My thread 1");
        MyThread m2 = new MyThread("My thread2");
    }
}
```

output:

Thread name: myThread1
Thread priority: Thread[myThread1, 10, main]
...
1
Thread name: myThread2
Thread priority: Thread[myThread2, 1, main]

Thread Synchronization Synchronization method

```
Public class SynThread {
    public static void main(String[] args) {
        Share S = new Share();
        MyThread M1 = new MyThread(S, "Thread 1");
        MyThread M2 = new MyThread(S, "Thread 2");
        MyThread M3 = new MyThread(S, "Thread 3"); }}
class myThread extends Thread { Codes for the thread }
class Share {
    public synchronized void docount(String str) {
        for (int i=0; i<5; i++) {
            S.O.P("Started." + str);
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
        }
    }
}
```

Output

Started Thread 1
.
.
.
Started Thread 3

Synchronized block:

A Block of code made as Synchronized

Syntax

```
class z
void. (int n) {
    Synchronized (this)
        normal code for loop
Main program
```

Static Thread synchronization: Program for tables

```
class Table {
    Synchronized static void PrintTable (int n) {
        for (int i=1; i<=10; i++) {
            S.O.P.(n*i);
            try {
                Thread.sleep (400);
            } catch (Exception e) { }
        }
    }
}
class MyThread1 extends Thread
{
    public void run()
    {
        Table.PrintTable (1);
    }
}
```

M T W T F S S

Page No.: YOUVA

Date:

Inter Thread communication

This mechanism allows synchronized threads to cooperate & uses methods from the object

wait(), notify(), notifyall()

notify() wake up single thread that is called wait()
notifyall() wakes up all thread that is called wait()

Code:

```
class customer {
    int amount=100;
    Synchronized void withdraw(int amount) {
        S.O.P("going to withdraw...");
        if(this.amount<amount){
            S.O.P("less balance");
            try{
                wait();
            }catch(exception e){}
        }
        this.amount-=amount;
        System.out.println("withdraw completed...");}
    Synchronized void deposit(int amount){
        S.O.P("going to deposit...");
        this.amount+=amount;
        S.O.P("completed");
        notify();
```

```
//Bank main function code
final customer c=new customer();
new thread(){
    public void run(){
        c.withdraw(15000);
    }
```

```
//
c.deposit(10000);
//
```

M T W T F S S
Page No.:
Date:
YOUVA

O/P:

going to withdraw
less balance, waiting for deposit
going to deposit
deposit completed
withdraw completed

Daemon Thread in Java

- A Daemon thread is a service provider (like Garbage collection) that runs in the background. Its life depends on user threads. the JVM terminates it automatically when all user threads die

Code:

```
public class TestDaemonThread1 extends Thread {
    public void run() {
        if (Thread.currentThread().isDaemon()) {
            S.O.P("Daemon thread work");
        } else {
            S.O.P("User thread work");
        }
    }
    public static void main(String[] args) {
        TestDaemonThread t1 = new TestDaemonThread1();
        t1.start();
    }
}
```

Output

daemon thread work
user thread work

M T W T F S S

Page No.: YOUVA

Date:

Interrupting Threads

* Interrupting a thread is a cooperative mechanism: It sets the threads interrupted status flag to true, requesting the thread to stop. The thread itself must check this status
* Method: Thread.interrupt()
* Behaviour in blocking state: If the thread is paused in a blocking method like sleep() or wait(), calling interrupt() will immediately throw an InterruptedException

Code:

```
public class ThreadInterruptDemo {
    public static void main(String[] args) {
        Thread sleepyThread = new Thread(() -> {
            try {
                S.O.P("Thread is going to sleep...");
                Thread.sleep(5000);
                S.O.P("Thread woke up normally
                        (this should not happen)");
            } catch (InterruptedException e) {
                S.O.P("Thread was successfully
                        interrupted while sleeping");
                S.O.P("Interrupted status after catch
                + Thread.currentThread().isInterrupted());
            }
        });
        try {
            Thread.sleep(1000);
            S.O.P("In Main thread is requesting interruption...");
            sleepyThread.interrupt();
        } catch (InterruptedException e) {
            //Main thread handler, ignored here)
        }
    }
```

Output:

Thread is going to sleep
Main thread is requesting interruption
Thread was successfully interrupted while sleeping
Interrupted status after catch: false

Methods of multithreading in Java

*) start — Initiates
*) currentThread
*) Run — triggers an action
*) isAlive — verify thread is alive
*) sleep — thread is suspended
*) yield — currently executing thread goes to standby mode
*) suspend — same like sleep
*) resume — resumes
*) interrupt — triggers an interruption of currently executing thread
*) destroy — destroys execution of group of threads
*) stop() — Stop the execution