OOPJ Notes Day-11 Session-1 Date: 09-05-2023

Important in Generic

- Generic method
- Type erasure

```
/*
class Demo<T extends Number>
{
    T a;

    void SetData(T a)
    {
        this.a=a;
    }
    void ShowData()
    {
        System.out.println("Value of a:"+a);
    }
}
*/
class Demo<Number>
{
    Number a;

    void SetData(Number a)
    {
        this.a=a;
    }
    void ShowData()
    {
        System.out.println("Value of a:"+a);
    }
}
public class TypeErDemo {

    public static void main(String[] args) {

        Demo d1=new Demo();

        d1.SetData("String");  //not valid

        d1.SetData(1121);

        d1.ShowData();
```

```
    }

}
```

- Bridge Method

```
class DemoBr<T>
{
    private T a;

    public T GetA()
    {
        return a;
    }

    void SetData(T a)
    {
        this.a=a;
    }
}

class DemoBr2<T> extends DemoBr<T>
{
    T b;
    DemoBr<T> b1=new DemoBr<T>();
    void SetData(T b, T a)
    {
        this.b=b;
        b1.SetData(a);
    }
    void ShowData()
    {
        System.out.println(b1.GetA()+" "+b);
    }
}
public class GenBrgMetDemo {

    public static void main(String[] args) {

        DemoBr2 d1=new DemoBr2();

        d1.SetData("Hello", "Hii");

        d1.ShowData();
    }
}
```

- Resriction on Generics
    1. Cannot Instantiate Generic Types with Primitive Types
    2. Cannot Declare Static Fields Whose Types are Type Parameters
    3. Cannot Use Casts or instanceof with Parameterized Types
    4. Cannot Create Arrays of Parameterized Types
    5. Cannot Create, Catch, or Throw Objects of Parameterized Types
    6. A class cannot have two overloaded methods that will have the same signature after type erasure.
- Fragile Base class problem
    - If we make changes in the body super class then we must recompile super class as well as all its sub classes. This problem is called as fragile base class problem.
    - We can solve fragile base class problem by defining super type as interface. ### Enum in Java

1. Declaration or Deffination of enum in java
    - inside of class deffination
    - outside of class deffination
2. Purpose of values() method in enum
3. Purpose of ordinal() method in enum ### Multi-Threading in java
4. Thread Concept
    - Thread Definition
        - Lightweight process is called as thread.
        - According to Java, thread is a seperate path of execution which runs independently Therad always resides in process.
        - If we want to utilize H/W resources(memory, cpu) efficiently then we should use thread.
        - If any application take help of single thread for the execution then such application is called as single threaded application.
        - If any application take help of multiple threads for the execution then such application is called as multi threaded application.
        - Thread is Non Java resource. It is also called unmanged resource.
5. Multi-Tasking
    - Term Singletasking and multitasking is always used in the context of Operating System An ability of operating system to execute single task at a time is called as Singletasking.
    - Example: MS DOS is singletasking operating system.
    - An ability of operating system to execute multiple task at a time is called as Multitasking. Example: MS Windows, Linuxm Mac OS etc.
    1. Process based Multi-Tasking
    2. Thread based Multi-Tasking
6. java as Multi-Threaded
    - When we start execution of Java application, JVM starts execution of main thread and Garbage collector. Due to these threads, every Java application is multithreaded.

- Main Thread
  - It is called as user thread / Non dameon thread.
  - Main thread is responsible for invoking main method.
  - In Java, priority of main thread is 5(Thread.NORM_PRIORITY).
- Garbage Collector
  - It is called as daemon thread / background thread.
  - Garbage collector is responsible for invoking finalize method and deallocating / releasing memory of unused objects.
  - Garbage collector is also called as finalizer.
  - In Java, priority of garbage collector is 8(Thread.NORM_PRIORITY + 3).

7. Multi-Threading in java
   1. Classes: Thread, ThreadGroup and ThreadLocal
   2. Interface: Runnable
   3. Enum: Thread.State
   4. Exceptions: IllegalThreadStateException, llegalMonitorStateException and InterruptedException
      - Runnable Interface
        - It is functional interface declared in java.lang package.
        - Method:
          - void run() //Program has to implements this fimction to create a new thread in java program.
          - To create thread, we can use Runnable interface.
      - Thread class
        - Thread is sub class of java.lang.Object class and it implements java.lang.Runnable interface.
        - Instance of Thread class is not a OS thread. Rather it represents OS thread.
        - JVM is responsible for mapping Thread instance with OS thread.
        - Thread.State is enum declared inside Thread class.
        - Fields:
          - public static final int MIN_PRIORITY //1
          - public static final int NORM_PRIORITY //5
          - public static final int MAX_PRIORITY //10
        - Constructors:
          - public Thread()
          - public Thread(String name)
          - public Thread(Runnable target)
          - public Thread(Runnable target, String name)
          - public Thread(ThreadGroup group, Runnable target, String name)
        - Methods:
          - public static Thread currentThread()
          - public final String getName()

- public final void setName(String name)
- public final int getPriority()
- public final void setPriority(int newPriority)
- public Thread.State getState()
- public final boolean isAlive()
- public final boolean isDaemon()
- public final void join() throws InterruptedException
- public final void setDaemon(boolean on)
- public static void sleep(long millis) throws InterruptedException
- public void start()
- public static void yield()

8. Thread creation using java.lang.Thread class and Runnable interface

```
class Demo implements Runnable
{

    @Override
    public void run() {
        for(int i=0;i<20;i++)
        {
        System.out.println("I am run of Demo");
        }
        //System.out.println(Thread.currentThread());
    }

}
class Demo1 implements Runnable
{

    @Override
    public void run() {
        for(int i=0; i<20;i++)
        {
        System.out.println("I am run of Demo1");
        }
        //System.out.println(Thread.currentThread());
    }

}
class Demo3 implements Runnable
{

    @Override
    public void run() {
        for(int i=0; i<20;i++)
```

```
        {
        System.out.println("I am run of Demo3");
        }
        //System.out.println(Thread.currentThread());
    }

}
public class MTThreadDemo{

    public static void main(String[] args) {

        Runnable r1=new Demo();

        Runnable r2=new Demo1();
        Runnable r3=new Demo3();

        Thread t1=new Thread(r1, "Demo Class Thread");
        Thread t2=new Thread(r2, "Demo-2 class Thread");
        Thread t3=new Thread(r3, "Demo-3 class Thread");

        t1.start();
        t2.start();
        t3.start();
        System.out.println("Now Running: "+Thread.activeCount());


    }

}
```

6. Thread life cycle or thread states

```
public class ThreadEnumDemo {

    public static void main(String[] args) {
        Thread t1=new Thread("My Thread");

        t1.start();

        System.out.println(t1.getName()+" "+t1.getPriority()+" "+t1.getState());

        System.out.println(Thread.currentThread());
    }

}
```

7. User Thread versus Daemon Thread

8. Thread termintation
9. Blocking calls in Thread
10. Race condition and synchronized keyword
11. Inter thread communication using wait,notify/notifyAll
12. Synchronization using consumer/producer ### try with resource

- External resources like File, Database con and N/W con to be in try block before its use.
- Will be discussed in Java I/O and JDBC, Java Socket Programming