1. **Write a java program to create an user defined exception called PayOutOfBoundsException. This exception is thrown when basicpay is not in between 10000 and 30000.**

**A.**

**import** java.util.\*;

**class** UserDefinedException {

**static** **void** validateInput(**int** BasicPay) **throws** PayOutOfBoundsException{

**if**(BasicPay<10000 || BasicPay>30000){

**throw** **new** PayOutOfBoundsException("Exception");

}

System.***out***.println("Valid Input");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter basic pay : ");

**int** BasicPay=sc.nextInt();

**try**{

*validateInput*(BasicPay);

}

**catch**(PayOutOfBoundsException e){

System.***out***.println("Caught Exception - Basic Pay should be in between 10000 and 30000");

}

}

}

**class** PayOutOfBoundsException **extends** Exception{

PayOutOfBoundsException(String s){

**super**(s);

}

}

**Output :**

Enter basic pay between 10000 and 30000 :

40000

Caught Exception - Basic Pay should be between 10000 and 30000

**2.Write a java program to create two threads which display a message every half second.**

**A. class** Thread1 **extends** Thread{

**public** **void** run( ) {

**try**{

**for**(**int** i = 1; i <= 5; i++) {

System.***out***.println("Thread1 is running");

Thread.*sleep*(500);

}

}

**catch**(InterruptedException e){

System.***out***.println("Interrupted.");

}

}

}

**class** Thread2 **extends** Thread {

**public** **void** run( ) {

**try**{

**for**(**int** i = 1; i <= 5; i++) {

System.***out***.println("Thread2 is running");

Thread.*sleep*(500);

}

}

**catch**(InterruptedException e){

System.***out***.println("Interrupted.");

}

}

}

**public** **class** Threads {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Thread1 a = **new** Thread1();

Thread2 b = **new** Thread2();

a.start();

b.start();

}

}

**Output :**

Thread1 is running

Thread2 is running

Thread1 is running

Thread2 is running

Thread1 is running

Thread2 is running

Thread2 is running

Thread1 is running

Thread2 is running

Thread1 is running

**3. Write a java program to implement interthread communication.**

**A. import** java.util.\*;

**class** Q

{

**int** n;

**boolean** valueset=**false**;

**public** **synchronized** **void** put(**int** n)

{

**while**(valueset)

{

**try**{

wait();

}

**catch**(InterruptedException e)

{

}

System.***out***.println("Put:"+n);

**this**.n=n;

valueset=**true**;

notify();

}

}

**public** **synchronized** **void** get()

{

**while**(!valueset)

{

**try**{wait();}

**catch**(InterruptedException e){}

}

System.***out***.println("Get:"+n);

valueset=**false**;

notify();

}

**class** Producer **implements** Runnable

{

Q q;

Producer(Q q)

{

**this**.q=q;

Thread t1 = **new** Thread(**this**,"Producer");

t1.start();

}

**public** **void** run()

{

**int** i=0;

**while**(**true**)

{

q.put(i++);

**try**{Thread.*sleep*(3000);}

**catch**(Exception e){}

}

}

}

**class** Consumer **implements** Runnable

{

Q q ;

Consumer(Q q)

{

**this**.q=q;

Thread t2=**new** Thread(**this**,"consumer");

t2.start();

}

**public** **void** run()

{

**while**(**true**)

{

q.get();

**try**{Thread.*sleep*(3000);}

**catch**(Exception e){}

}

}

}

**static** **class** InterthreadCommunication

{

**public** **static** **void** main(String[] args)

{

Q q=**new** Q();

**new** Producer(q);

**new** Consumer(q);

}

}

}

**Output :**

Put : 0

Get : 0

Put : 1

Get : 1

Put : 2

Get : 2

Put : 3

Get : 3

Put : 4

Get : 4

Put : 5

Get : 5

Put : 6

Get : 6

Put : 7

Get : 7

Put : 8

Get : 8

Put : 9

Get : 9

Put : 10

Get : 10

**4.Write a java program to implement Thread Synchronization.**

**A.**

**class** Table{

**synchronized** **void** printTable(**int** n){

**for**(**int** i=1;i<=10;i++){

System.***out***.println(n\*i);

**try**{

Thread.*sleep*(500);

}**catch**(Exception e){System.***out***.println(e);}

}

}

}

**public** **class** SynchronizedThread{

**public** **static** **void** main(String args[]){

**final** Table obj = **new** Table();

Thread t1=**new** Thread(){

**public** **void** run(){

obj.printTable(3);

}

};

Thread t2=**new** Thread(){

**public** **void** run(){

obj.printTable(8);

}

};

t1.start();

t2.start();

}

}

**Output :**

3

6

9

12

15

18

21

24

27

30

8

16

24

32

40

48

56

64

72

80

**5.Write a java program to implement Generic Class,Generic Method and Generic Constructor.**

**A. public** **class** Generics <T1,T2> {

// generic method printArray

**public** **static** < E > **void** printArray( E[] inputArray ) {

// Display array elements

**for**(E element : inputArray) {

System.***out***.printf("%s ", element);

}

System.***out***.println();

}

**public** **static** **void** main(String args[]) {

// Create arrays of Integer, Double and Character

Integer[] intArray = { 32, 56, 76, 12, 49 };

Double[] doubleArray = { 15.6, 12.5, 45.3, 15.9 };

Character[] charArray = { 'J', 'A', 'V', 'A' };

System.***out***.println("Array integerArray contains:");

*printArray*(intArray); // pass an Integer array

System.***out***.println("\nArray doubleArray contains:");

*printArray*(doubleArray); // pass a Double array

System.***out***.println("\nArray characterArray contains:");

*printArray*(charArray); // pass a Character array

}

}

**Output :**

Array integerArray contains:

32 56 76 12 49

Array doubleArray contains:

15.6 12.5 45.3 15.9

Array characterArray contains:

J A V A

**6.Write a java program to count no of vowels in a given file.**

**A. public** **class** CountVowels {

**public** **static** **void** main(String[] args) {

**int** vCount = 0;

String str = "I am learning java";

str = str.toLowerCase();

**for**(**int** i = 0; i < str.length(); i++) {

//Checks whether a character is a vowel

**if**(str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i' || str.charAt(i) == 'o' || str.charAt(i) == 'u') {

//Increments the vowel counter

vCount++;

}

}

System.***out***.println("Number of vowels: " + vCount);

}

}

**Output :**

Number of vowels: 7

**7. Write a java program to implement autoboxing and unboxing.**

**A.**

**public** **class** Auto {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**double** a = 5.475;

Double doubleobj=a;//Auto-boxing

System.***out***.println("Wrapper class object :"+doubleobj);

**double** c = doubleobj; //Un-boxing

System.***out***.println("Primitive value :"+c);

}

}

**Output :**

Wrapper class object :5.475

Primitive value :5.475

**8. Write a java program to copy a file.**

**A.**

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**class** Main {

**public** **static** **void** main(String[] args) {

**byte**[] array = **new** **byte**[50];

**try** {

FileInputStream sourceFile = **new** FileInputStream("input.txt");

FileOutputStream destFile = **new** FileOutputStream("newFile");

// reads all data from input.txt

sourceFile.read(array);

// writes all data to newFile

destFile.write(array);

System.***out***.println("The input.txt file is copied to newFile.");

// closes the stream

sourceFile.close();

destFile.close();

}

**catch** (Exception e) {

e.getStackTrace();

}

}

}

**Output :**

The input.txt file is copied to new file

**9. Write a java program to implement Stack using Generic class.**

**A.**

**import** java.util.\*;

**public** **class** StackUsingGen <T> {

**private** ArrayList<T> stack = **new** ArrayList<T> ();

**private** **int** top = -1;

**public** **int** size () { **return** top+1; }

**public** **void** push (T item) {

stack.add(++top, item);

}

**public** **boolean** isEmpty()

{

**return** (top == -1);

}

**public** T pop () {

**return** stack.remove (top--);

}

**public** **static** **void** main (String[] args) {

StackUsingGen<Integer> s = **new** StackUsingGen<Integer> ();

s.push(17);

s.push(9);

s.push(8);

System.***out***.println(s.size());

**int** a = s.pop ();

System.***out***.println(a);

**int** b = s.pop ();

System.***out***.println(b);

System.***out***.println(s.isEmpty());

**int** c = s.pop ();

System.***out***.println(c);

System.***out***.println(s.isEmpty());

}

}

**Output :**

3

8

9

false

17

True

**10. Write java program to swap two values using generic method.**

**A. public** **class** GenSwap <Integer>

{

**public** GenSwap(**int** a, **int** b) {

// **TODO** Auto-generated constructor stub

a=a+b;

b=a-b;

a=a-b;

}

**public** **int** getFirst(**int** a){

**return** a;

}

**public** **int** getsecond(**int** b){

**return** b;

}

//public String toString() { return "(" + first + ", " + second + ")"; }

**public** **static** **void** main(String[] args){

**int** a = 10;

**int** b = 6;

GenSwap<Integer> s = **new** GenSwap<Integer>(a,b);

System.***out***.println(s.getFirst());

System.***out***.println(s.getSecond());

}

}

**Output :**

6

10

**1.What is thread? A.** A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

**2. Write the difference between multithreading and multitasking. A. Multithreading:**

* Multithreading is a conceptual programming paradigm where a process is divided into a number of sub-processes called as threads. Each thread is independent and has its own path of execution with enabled inter thread communication.
* In multithreading, CPU is provided in order to execute many threads from a process at a time, and in multithreading, process creation is performed according to cost.
* Unlike multitasking, multithreading provides the same memory and resources to the processes for execution.

**Multitasking:**

* Multitasking is when a CPU is provided to execute multiple tasks at a time.
* Multitasking involves often CPU switching between the tasks, so that users can collaborate with each program together.
* Unlike multithreading, In multitasking, the processes share separate memory and resources.
* As multitasking involves CPU switching between the tasks rapidly, So the little time is needed in order to switch from the one user to next.

**3. What is Enumeration? A.** An **enumeration** or **enum** is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

To create an **enum**, use the **enum** keyword (instead of class or interface), and separate the constants with a comma.

**4. What is autoboxing? A.** The automatic conversion of primitive data types into its equivalent Wrapper type is known as auto-boxing.

**5. What is wrapper class? A.** A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types.

**6. what is transient modifier? A.** An instance variable is marked transient to indicate the JVM to skip the particular variable when serializing the object containing it.

This modifier is included in the statement that creates the variable, preceding the class or data type of the variable.

**Example :**

public **transient** int a = 100; //will not persist public int b = 50; //will persist

**7. What is Generic class?Write the syntax of generic class. A.** A class is said to be Generic if it declares one or more type variables. These variable types are known as the type parameters of the [Java Class](https://www.edureka.co/blog/java-tutorial/#obj). We use angle brackets (<>) to specify the type parameter.

**Syntax :**

class GenericClass <x>{

private T x; //T stands for "Type"

public void set (T x) { this.x = x; }

public T get ( ) { return x; }

}

**8. What is stream? A.** The Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

**9.What is predefined stream? A.** A **Java** program has three **predefined streams** available. These **streams** are defined in the **java**.lang package's System class. They are :

**1) System.out:**standard output stream

**2) System.in:**standard input stream

**3) System.err:**standard error stream

These fields are declared as **public**, **final**, and **static** within **System**. This means that they can be used by any other part of your program and without reference to a specific **System** object.

**10.What is multithreading? A.** Multithreading is a conceptual programming paradigm where a process is divided into a number of sub-processes called as threads.

**11. What is the use of toString()? A.** The **toString()** method is used to get string representation of an object.

**Syntax of Object class toString() method:**

Public String toString() {

Return getclass().getName() + “@” + Integer.toHexString(hashCode());

}

**12.What is deadlock? A.** Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

**13. Write inter thread communication methods. A.** Inter-thread communication (Cooperation) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

**1)wait() method** Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

**Syntax :**

**Method Description**

public final void wait()throws InterruptedException waits until object is notified.

public final void wait(long timeout)throws InterruptedException waits for the specified amount of time.

**2)notify() method** Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

**Syntax:**

public final void notify()

**3)notifyAll() method** Wakes up all threads that are waiting on this object's monitor.

**Syntax:**

public final void notifyAll()

**14. Write the difference between Checked and Unchecked exception. A.**

**1)Checked Exception**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions. Checked exceptions are checked at compile-time.

**Ex:** IOException, SQLException etc.

**2)Unchecked Exception** The classes which inherit RuntimeException are known as unchecked exceptions.Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**Ex:** ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

**15. What is thread synchronization? A.** Thread synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

**General Syntax:**

synchronized (object) {

//statement to be executed

}