Explain the use of keywords "try", "catch", "throw", "throws" and finally with proper java program.

1) "try" block

- Java try block is used to enclose the code that might throw an exception.
- It must be used within the method.
- Java try block must be followed by **either catch or finally block.**

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

2) "catch" block

- Java catch block is used to handle the Exception.
- **It must be used after the try block only.**
- You can use multiple catch block with a single try.

Problem without exception handling

- Let's try to understand the problem if we don't use try-catch block.

```
class TestTryCatch
{
 public static void main(String args[]){
    int data=50/0;//may throw exception
    System.out.println("rest of the code...");
  }
}
```

Output

```
Exception in thread main
java.lang.ArithmeticException:/ by zero
```

- rest of the code is not executed
- There can be 100 lines of code after exception.
- So all the code after exception will not be executed.

Solution by exception handling

- Let's see the solution of above problem by java try-catch block.

```
class Testtrycatch2
{
     public static void main(String args[])
     {
         try{
             int data=50/0;
         }catch(ArithmeticException e){
         System.out.println(e);
         }
         System.out.println("rest of the code...");
     }
}
```

3) "throw" keyword

- The Java throw keyword is used to **explicitly throw** an exception.
- We can throw either checked or uncheked exception in java by throw keyword.
- The throw keyword is **mainly used to throw custom exception.**
- The **syntax** of java throw keyword is given below.

```
    throw exception;
```

- Let's see the example of throw IOException.

```
throw new IOException("sorry device err");
```

Example

```
class TestThrow1{
 public static void main(String args[])
{
  int age=13;
  if(age<18)
    throw new ArithmeticException("not valid");
 else
    System.out.println("welcome to vote");

    System.out.println("rest of the code...");
 }
}
```

Output:

```
Exception in thread main
java.lang.ArithmeticException:not valid
```

4) "throws" keyword

- The **"throws keyword"** is used to **declare** an exception.
- It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Exception Handling is mainly used to handle the checked exceptions.

Syntax of java throws

```
return_type method_name()
throws exception_class_name
{
...
}
```

- **We can declare multiple exception e.g.**

```
public void method()
throws IOException,SQLException
{}
```

5) "finally" block

- **finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- **finally block is always executed whether exception is handled or not.**
- finally block **must be followed by try** or catch block.

```
class TestFinallyBlock2{
 public static void main(String args[])
 {
 try{
        int data=25/0;
        System.out.println(data);
 }
 catch(ArithmeticException e){
        System.out.println(e);
 }
 finally{
System.out.println("It is always executed");
 }
 System.out.println("rest of the code...");
 }
}
```

- **Points to remember**
- Try must have either catch or finally.
- At a time only one Exception is occured and at a time only one catch block is executed.
- For each try block there can be zero or more catch blocks, but only one finally block.
- The finally block will not be executed if program exits by calling System.exit().

Differentiate the Following

throw	throws
throw keyword is used to throw an exception explicitly.	throws clause **is used to declare an exception**

throw keyword can be **used inside method** or static initializer block
 	
static{
try {
throw new Exception("Not able to initialized");
} catch (Exception ex) {}
}
	throws keyword cannot be used anywhere except **method signature**
	
	void display() throws IOException,SocketException
	{
	....
	}

If we see syntax wise than throw is followed by an instance variable of Throwable	**throws** is followed by exception class names.
The keyword throw is used inside method body to invoke an exception	**throws clause** is used in method declaration (signature).
By using throw keyword in java you cannot throw more than one exception	using **throws** you can declare multiple exceptions
Example **throw new ArithmeticException()**	Example **throws IOException, SocketException**
throw keyword can also be used to break a switch statement without using break keyword	throws cannot be used inside method or switch case.

Checked Exception (Compile Time Exception)	Unchecked Exception (Runtime Exception)
The exceptions that are checked at **compile time** are called checked exceptions,	The exceptions that **are not checked at compile time** are called unchecked exceptions,
Checked exceptions **must be explicitly caught** or to be thrown	Unchecked exceptions do not have this requirement. They **don't have to be caught** or declared thrown.
In Exception hierarchy all classes that **extends Exception class except RuntimeException** comes under checked exception category.	classes that **extends RuntimeException comes** under unchecked exceptions.
Some common examples of Checked Exception • **IOException** • **ParseException** • **InterruptedException**	Some common example of Unchecked Exception • **NullPointerException** • **ArithmeticException** • **ArrayIndexOutOfBoundsException**
Checked Exception need to be handled by caller	Unchecked Exception don't

Q.3. Explain Custom Exception in detail. OR Explain User Defined Exception in detail.

- If you are creating your own Exception that is known as **custom exception or user-defined exception.**
- Java custom exceptions are used to customize the exception according to user need.
- By the help of custom exception, you can have your own exception and message.

Example

```
class InvalidAgeException extends Exception
{
        InvalidAgeException(String s)
        {
        super(s);
        }
}
class TestCustomException1
{
  public static void main(String args[])
throws InvalidAgeException
  {
   int age=13;
   if(age<18)
   throw new InvalidAgeException("not valid");
   else
    System.out.println("welcome to vote");
 }
}
```

- **To Create Checked-Custom Exception**
  - Create class which extends from Exception
- **To Create Unchecked-Custom Exception**
  - Create class which extends from RuntimeException

Q.4. Explain Nested Class with Example

1. Top Level Nested Class
2. Inner Class
3. Local Class
4. Anonymous Class

1) TOP-LEVEL NESTED CLASS

- When we declare static member class it is known as **Top Level Nested Class.**

```
class A
{
  static class B // Top Level Nested Class
   {
   }
}
```

2) INNER CLASS

- When we declare non-static member class inside class is known as inner class.

```
class A{
        class B // Inner Class
        {

        }
}
```

3) LOCAL INNER CLASS

- When we declare class inside method, constructors or block is known as Local class.

```
class A{
        void display(){
                class B // Local class
                {
                }
        }
}
```

4) ANONYMOUS INNER CLASS

- when we declare class without name it is known anonymous.
- It is defined and created at same place.

e.g.

```
new FilenameFilter()
{
    public boolean accept(File path,String name)
    {
        return name.endsWith(".java");
    }
}
```

TOP LEVEL NESTED CLASS Example

```
class Test
{
	private static int x=1;
	static class A
	{
		private static int y=100;
		public static int getZ()
		{
			return B.z+x;
		}
	}
	static class B
	{
		private static int z=200;
		public static int getY()
		{
			return A.y;
		}
	}
}
class TestDemo
{
	public static void main(String []args)
	{
		Test t = new Test();
		System.out.println(Test.A.getZ());
		System.out.println(Test.B.getY());
	}
}
```

INNER CLASS EXAMPLE

```
class Test
{
	private int i=100;
	class A
	{
		private int i=200;
		void display()
		{
		    int i=300;
		    System.out.println(i);
		    System.out.println(this.i);
		    System.out.println(Test.this.i);
		}
	}
}
```

LOCAL INNER CLASS Example

```
class Test
{
	Test()
	{
		class A // Local Class
		{
			int i=100;
			A()
			{
			System.out.println(i);
			}
		}
		System.out.println(new A().i);
	}
}
```

Anonymous Inner Class Example

```
class Test
{
    public void display()
    {
        System.out.println("Hello From Test");
    }
}
class TestDemo
{
	public static void main(String []args)
	{
		Test t = new Test()
		{
		    public void display()
		    {
System.out.println("Hello From Anonymous");
		    }
		};
		t.display();
	}
}
```

OR

```
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
       System.exit(0);
   }
}
```

Q.5. Explain enum nested type in detail.

- When we have fixed no of instance in this situation we can use enum.
- super class of enum is abstract class **Enum.**
- enum is reference datatype and it is final
    - so, we cannot create sub-class from enum
- enum instance variables are static, final
- enum constructors are always private
- we can declare methods inside enum.

Methods of Enum class
==========
String name() - It returns name of an instance
int ordinal() - It returns an index of an instance.
enum[] values() - It returns all the instances of enum type in an array.

Example of enum type

```
// enum showing Mobile prices
enum Mobile
{
  Samsung(400), Nokia(250),Motorola(325);

  int p;
  Mobile(int p) {
    this.p = p;
  }
  int showPrice() {
    return p;
  }
}

class EnumDemo {
  public static void main(String args[])
 {
   System.out.println("CellPhone List:");
   for(Mobile m : Mobile.values())
   {
     System.out.println(m + " costs " +
m.showPrice() + " dollars");
   }

 Mobile ret = Mobile.Nokia;
 System.out.println("Ordinal:" + ret.ordinal());
 System.out.println("Name: " +ret.name());
 System.out.println("Price: " + ret.showPrice());
  }
}
```

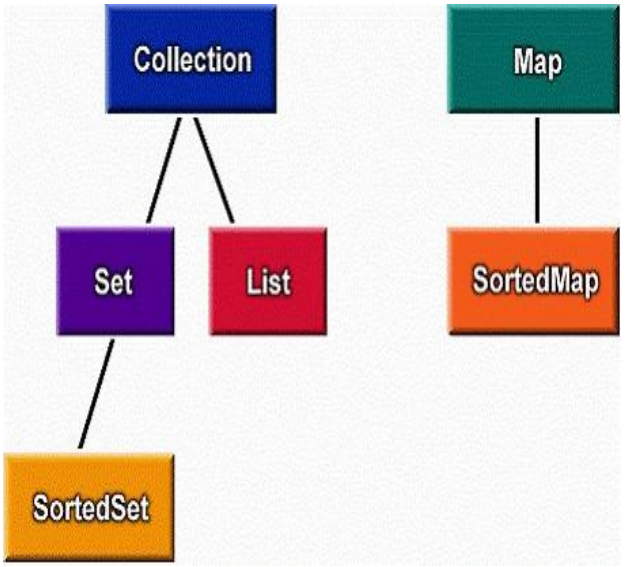
Q.6.What is Collection in Java?

- **Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Collections.
- Collection simply means a single unit of objects.
- Collection framework provides many interfaces (Set, List, Map etc.) and classes (ArrayList, Vector, LinkedList, HashSet, TreeSet etc).

- **What is Collection in java**
  - Collection represents a single unit of objects i.e. a group.
- **What is framework in java**
  - provides readymade architecture.
  - represents set of classes and interface.
  - is optional.
- **What is Collection framework**
  - Collection framework represents a unified architecture for storing and manipulating group of object.
  - It has Interfaces and its implementations.
- Collection Enables you to work with groups of objects; it is at the top of the collections hierarchy
- List Extends **Collection** to handle sequences (lists of objects)
- Set Extends **Collection** to handle sets, which must contain unique elements
- SortedSet Extends **Set** to handle sorted sets

Collection framework has following important interfaces.

- **Collection** — the root of the collection hierarchy.
- **Set** — a collection that cannot contain duplicate elements.
- **List** — an ordered collection (sometimes called a *sequence*) .
- **SortedSet** — a Set that maintains its elements in ascending order.



Common Methods of Collection

- Basic operations
  - add
  - remove
  - size
  - clear
  - contains
  - isEmpty
  - iterator
- Array operations
  - toArray()
  - toArray(Object[] o)
- Bulk operations
  - addAll
  - removeAll
  - retainAll
  - containsAll

Q.7. Explain List and Set Interface of Collection.

List interface

- A List is an ordered collection. (sometimes called a *sequence*).
- Lists can contain duplicate elements.
- The user can access elements by their integer index (position).
- List interface extends from Collection
- It has all the methods of Collection and has additional methods for index operations.
- List has additional methods for add, remove, and modified objects based on index.
- List allows duplicate elements...
    - void add(int index, Object o)
    - Object set(int index,Object o)
    - Object get(int index)
    - Object remove(int index)
    - int indexOf(Object o)
    - List subList(int start,int end)
    - ListIterator listIterator()

Set Interface

- A Set is a collection that cannot contain duplicate elements.
- As you might expect, this interface models the mathematical *set* abstraction.
- Set is an interface extends from Collection.
- It has no additional methods, it has all the methods of Collection.
- It is collection of an unique elements only.
- HashSet is a class which implements Set interface.

Sorted Set Interface

- SortedSet interface extends from Set interface.
- It is used to arrange unique set of collection in ascending order.
- TreeSet is a class which implements SortedSet interface
- Several additional operations are provided to take advantage of the ordering.

Example of List using Iterator

```
import java.util.*;
class IteratorDemo
{
        public static void main(String []args)
        {
                List c = new ArrayList();
                c.add(10);
                c.add(20);
                c.add(1,30);
                c.add(40);
                System.out.println(c);

                Iterator i = c.iterator();
                while(i.hasNext())
                {
                        Object o = i.next();
                        if(o==(Integer)40)
                           i.remove();
                        else
                          System.out.println(o);
                }
                System.out.println(c);
        }
}
```

Q.8. Explain Map interface with Example.

Map Interface :

- It is collection of key and value.
- It maintains value based on key.
- It is collection of unique key.
- It has key and value both are Object.

Methods
=======

```
Object put(Object key,Object value)
Object get(Object Key)
Object remove(Object Key)
boolean isEmpty()
void clear()
Set entrySet()
Collection values()
void putAll(Map m)
```

Map.Entry Interface
==========

```
Object getKey()
Object getValue()
Object setValue(Object o)
```

Example:

```
import java.util.*;
class Test
{
	public static void main(String []args)
	{
	Map<Integer,String> m =
		new HashMap<Integer,String>();

		m.put(1,"MCA");
		m.put(2,"BCA");
		m.put(3,"BSCIT");
		m.put(2,"MSCIT");

	Set s = m.entrySet();
	Iterator i = s.iterator();
	while(i.hasNext())
	{
	Map.Entry me = (Map.Entry)i.next();
	System.out.print(me.getKey()+" ");
	System.out.println(me.getValue());
		}
	}
}
```

Explain Generic Collection in Java.

- A class or an interface is generic if it has one or more type variable.
- Type variable are delimited by angle brackets and follow the class (or the interface) name:

```
public interface List<T> extends Collection<T>
{
}
```

Example

```
List<String> str=new ArrayList<String>();
str.add("Hello ");
str.add("World ");
```

Generic Wild-Card

- ✓ Collection : It is a collection of an object
- ✓ Collection<Integer> : It is a collection of an Integer Type Only
- ✓ Collection<Person> : It is a collection of a Person Type.
- ✓ Collection<?> : It is a collection of any kind of an object.
- ✓ Collection<? extends Person> : Person + Subclass of Person
- ✓ Collection<? super Allrounder> : Any class which are super class of Allrounder.

Example

```
import java.util.*;
class TestDemo
{
  public static void printCollection
    (Collection<? extends Number> c)
  {
        for(Object o : c)
                System.out.println(o);
   }
   public static void main(String []args)
   {
         Collection<Number> c = new ArrayList();
           c.add(10l);
           c.add(20.2);
           c.add(30.3f);
           c.add(40);
           TestDemo.printCollection(c);
   }
}
```

Differentiate the Following

Set	List
It is a collection of Unique elements.	It may contain duplicate elements.
It is a distinct list of elements which is unordered.	It is an ordered sequence of elements.
No new additional methods other than Collection.	New methods other than Collection are defined in List.
Can use only Iterator to traverse a Set.	Can use both Iterator and ListIterator to traverse a list
Set does not allow NULL values.	List interface allows NULL values.
Subclass of Set are • HashSet (Unordered) • TreeSet from SortedSet (Ordered)	Subclss of List • ArrayList • LinkedList • Vector
Set s = new HashSet()	List l = new ArrayList();

Iterator Vs ListIterator

Iterator	ListIterator
It inherits from Iterable	It inherits from Iterator
We can use Iterator to traverse a Set or List or a Map.	But ListIterator can only be used to traverse a List.
Using Iterator we can retrieve the elements from Collection in forward direction only,	ListIterator allows us to traverse in both directions.
We can remove only current element.	We can remove, add, and replace any element as per index.
We cannot get any index using Iterator	We can get previousIndex() and nextIndex() using ListIterator

Collection c = new ArrayList()
 	
c.add(10);
c.add(20);
c.add(30);
 	
Iterator i = c.iterator();
while(i.hasNext())
System.out.println(i.next());
	List l = new ArrayList()
	
	l.add(10);
	l.add(0,20);
	l.add(30);
	
	ListIterator li = l.listIterator();
	while(li.hasNext()){
	System.out.println(li.next());
	System.out.println(li.nextIndex());
	}

Q.9. Explain Pattern and Matcher of Regular Expression with Example.

- A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- They can be used to search, edit, or manipulate text and data.
- The **java.util.regex** package primarily consists of the following classes:
- **Pattern Class:**
  - A Pattern object is a compiled representation of a regular expression.
  - The Pattern class provides no public constructors.
  - To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object.
  - These methods accept a regular expression as the first argument.

Methods

Pattern compile(String str)
String[] split(String)
Matcher matcher(String)

Pattern p = Pattern.compile("pattern");

- **Matcher Class:**
  - A Matcher object is the engine that interprets the pattern and performs match operations against an input string.
  - Like the Pattern class, Matcher defines no public constructors.
  - You obtain a Matcher object by invoking the matcher method on a Pattern object.

Methods

boolean matches()
boolean find()
int start()
String group()

Matcher m = p.matcher("matching string");

+ : One or more
*** :** Zero or more
. : Exactly One
? : Zero or one

Example :

Regular Expression to valid Mobile No

```
import java.util.regex.*;
class TestDemo {

  public static void main(String args[])
  {
    Pattern p = Pattern.compile("[0-9]{10}");
    Matcher m = p.matcher(args[0]);
    System.out.println(m.matches());
  }
}
```

Output:

If You pass 10 digit at command line then it will print true otherwise it will print false.

Regular Expression to valid Password

```
import java.util.regex.*;
class Test{
        public static void main(String[] args) {
                String input = args[0];
                Pattern p =
Pattern.compile("((?=.*[0-9])(?=.*[a-z])(?=.*[A-
Z])(?=.*[@%#$]).{8,20})");
                Matcher m = p.matcher(input);
                System.out.println(m.matches());
        }
}
```

- **Regular Expression for Username Validation**

```
String pattern = "[a-z0-9_-]{3,15}";
Pattern p = Pattern.compile(pattern);
Matcher m = p.matcher(args[0]);
System.out.println(m.matches())
```

Q.10. Varargs in Java

- Varargs - Variable no of arguments
- It must be the last argument of function
- It is denoted by ... (triple dot)
- It is same as printf function in C Lang

Example of inbuilt function

- printf("Hello")
- printf("%d",150);
- printf("%s %d","Atmiya",150);

Example of user defined vaargs

```
class Test{
        void display(Object ... oa){
          System.out.println("Display Calls");
          for(Object o : oa){
                System.out.print(o+"\t");
           }
           System.out.println();
        }
        public static void main(String[] args) {
           Test t = new Test();
           t.display();
           t.display(10,20);
           t.display(10,20,"Atmiya");
          t.display("Atmiya University",10,20,"MCA");
        }
}
```

Sort any data types using Varargs and Arrays

```
import java.util.*;
class Test{
        void display(Object ... oa){
                Arrays.sort(oa);
                for(Object o : oa){
                        System.out.print(o+"\t");
                }
                System.out.println();
        }
        public static void main(String[] args) {
                Test t = new Test();
                t.display(32,56,12,65,87,21,45,23);
             t.display("MCA","University","Atmiya");
                t.display('A','N','A','N','D', '  ', 'T',
                'A''N''K');
        }
}
```

Program using Map

Create a "student" class. Student has members stdid, name, grade. Create HashMap for student objects . Write an application which adds student objects in HashMap and displays all students objects from HashMap with the help of Collection interface. Write appropriate constructors and methods in student class.

```
import java.util.*;
class Student
{
        private int stdid;
        private String name;
        private String grade;
        public Student() {}
        public Student(int stdid, String name,
String grade) {
                this.stdid = stdid;
                this.name = name;
                this.grade = grade;
        }
        @Override
        public String toString() {
                return stdid+" "+name+" "+grade;
        }
}

class StudentMap {
        public static void main(String []args){
        Map<Integer,Student> m =
                new HashMap<Integer,Student>();

      m.put(1,new Student(101,"Ram","AB"));
      m.put(2,new Student(102,"Laxman","BB"));
      m.put(3,new Student(103,"Bharat","BC"));
      m.put(4,new Student(104,"Sitaji","AA"));
      System.out.println(m);

         Set c = m.entrySet();
         Iterator i = c.iterator();
         while(i.hasNext()){
                Student p = (Student)i.next();
                System.out.println(p);
               System.out.println(p.grade);
        }
  }
}
```