1. **Suppose I have an enum defined say Subjects with Values: DBMS, DSA, NETWORKING,OPERATING SYSTEMS. a) How do I access the index of the values present in the enums. b) If while accessing the enums I pass an argument or say value which is not present in the enum then what kind of exception will be thrown?**

**c) Can you demonstrate the same using a Java program?**

**Answer**:

* **(a)**
* Using ordinal() method of Enum class we can get the **index** of any element from enum .

**Program:**

enum Subjects{

DBMS, DSA, NETWORKING, OPERATING\_SYSTEMS;

}

public class EnumTest{

public static void main(String args[]){

Subjects [] subjects = Subjects.values();

for(Subjects s : subjects){

System.out.println(“Index of ”+s+” is ”+s.ordinal());

}

}

}

**Output:**

Index of DBMS is **0**

Index of DSA is **1**

Index of NETWORKING is **2**

Index of OPERATING SYSTEM is **3**

* **(b)**
* It will throw **java.lang.IllegalArgumentException: No enum constant**

**Program:**

enum Subjects1{

DBMS, DSA, NETWORKING, OPERATING\_SYSTEMS;

}

public class EnumTest2{

public static void main(String args[]){

//we take one subject which is not available in enum in order to

//throw an exception

//it will throw **java.lang.IllegalArgumentException**

String str = “JAVA”;

System.out.println(Subjects1.valueOf(str));

}

}

**Output:**

Exception in thread "main" java.lang.IllegalArgumentException: No enum constant Subjects1.JAVA

at java.base/java.lang.Enum.valueOf(Enum.java:273)

at Subjects1.valueOf(EnumTest2.java:1)

at EnumTest2.main(EnumTest2.java:11)

**2. Will the following method compile? If not, why?**

*public static void print(List<? extends Number> list) {*

*for (Number n : list)*

*System.out.print(n + " ");*

*System.out.println();*

*}*

**Answer:**

* **Yes**, the above code will be compiled successfully and basically this is an upper-bounded wild card, that means the above list can store the objects of the class that extends the **Number** class only(such as Integer, Double etc) and no other types are allowed(which not extends the **Number** class).
* If we use the class type that does not extend the **Number** class will cause a compilation error.

**3. If the compiler erases all type parameters at compile time, why should you use generics?**

**Answer:**

* You should use **Generics** because :
* It provides **Type-Safety** that means we can hold only single type of objects and it does not allowed us to store other objects.
* **Generics forces the errors to appear at compile-time** rather than appearing on runtime. It is far better to handle the program at compile-time than runtime.
* Manual **Type-casting is not required** , there is no need to type-cast the object. Before Generics, we need to type cast.
* Generic **enable us to implements Generic algorithm.**

**4. What is the following class converted to after type erasure?**

*public class Pair <K, V> {*

*public Pair(K key, V value) {*

*this.key = key;*

*this.value = value;*

*}*

*public K getKey() { return key; }*

*public V getValue() { return value; }*

*public void setKey(K key) { this.key = key; }*

*public void setValue(V value) { this.value = value; }*

*private K key;*

*private V value;*

*}*

**Answer:**

=> During the Type-erasure process, the Java compiler erases all type parameters and replace each with first bound if the type parameter is bouded

public class Pair {

public Pair(Object key, Object value) {

this.key = key;

this.value = value;

}

public Object getKey() { return key; }

public Object getValue() { return value; }

public void setKey(Object key) { this.key = key; }

public void setValue(Object value) { this.value = value; }

private Object key;

private Object value;

}

***5. Look at the following code snippet and select the correct option:***

*class Test extends Exception { }*

*class Main {*

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

*try {*

*throw new Test();*

*} catch(Test t) {*

*System.out.println("Got the Test Exception");*

*}*

*finally {*

*System.out.println("Inside finally block ");*

*}*

*}*

*}*

*Options:*

*a) Got the Test Exception*

*Inside finally block*

*b) Got the Test Exception*

*c) Inside finally block*

*d) Compiler Error*

**Answer:**

**a) Got the Test Exception**

**Inside finally block**

**6. What will be the output of the following code:**

*class Test {*

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

*try {*

*int a[]= {1, 2, 3, 4};*

*for (int i = 1; i <= 4; i++) {*

*System.out.println ("a[" + i + "]=" + a[i] + "\n");*

*}*

*}*

*catch (Exception e) {*

*System.out.println ("error = " + e);*

*}*

*catch(ArrayIndexOutOfBoundsException e){*

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

*}*

*}*

*}*

*a) Compiler error*

*b) Run time error*

*c) ArrayIndexOutOfBoundsException*

*d) Error Code is printed*

*e) Array is printed*

**Answer:**

**a) Compiler error:** Because in first catch block all Exceptions including ArrayIndexOutOfBoundsExceptionare caught so we don’t need to write second catch block, otherwise it will throw compilation error.

In this case, it also throws compilation error