Object Oriented Concept

I.Introduction

II Pillars Of OOPs

2.4 Abstraction

III Generics

- Generics means parameterized types.
- The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces.
- **Using Generics, it is possible to create classes that work with different data types**.
- An entity such as class, interface, or method that operates on a parameterized type is a generic entity.
- Generics were introduced in Java

Ex:
public <T> List<T> **fromArrayToList**(T[] a) { return Arrays.stream(a).collect(Collectors.toList()); }

Type Parameters in Java Generics

The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

T – Type
E – Element
K – Key
N – Number
V – Value

Wildcards With Generics

A wildcard type represents an unknown type.

public static void paintAllBuildings(List**<? extends Building>** buildings) {
    ...
}

Bounded Generics

Bounded means “restricted,” and we can restrict the types that a method accepts.
Ex: *public <**T extends Number**> List<T> fromArrayToList(T[] a) {*
    ...
}

Multiple bounds:

<T extends Number & Comparable>

If one of the types that are extended by T is a class (e.g. Number), we have to put it first in the list of bounds. Otherwise, it will cause a compile-time error.

- We can use wildcards with bounds in three ways:
    - **Unbounded Wildcards: List<?>** – represents a list of any type
    - **Upper Bounded Wildcards**: **List<? extends Number**> – represents a list of Number or its subtypes (for instance, Double or Integer).
    - **Lower Bounded Wildcards: List<? super Integer**> – represents a list of Integer or its supertypes, Number, and Object
- We can bound type parameters in two ways:
    - **Unbounded Type Parameter: List<T>** represents a list of type T
    - **Bounded Type Parameter: List<T extends Number & Comparable>** represents a list of Number or its subtypes such as Integer and Double that implement the Comparable interface

Difference b/w type parameter and wild card

- We can't use **type parameters with the lower bound.** Furthermore, type parameters can have **multiple bounds, while wildcards can't.**
- If a **type parameter appears only once in the method declaration,** we should consider **replacing it with a wildcard.**

Type Erasure

- Generics were added to Java to **ensure type safety.**
- **And to ensure that generics won't cause overhead at runtime, the compiler applies a process called type erasure on generics at compile time.**
- Type erasure removes all type parameters and replaces them with their bounds or with Object if the type parameter is unbounded.
- This way, the bytecode after compilation contains only normal classes, interfaces and methods, ensuring that no new types are produced.
- Proper casting is applied as well to the Object type at compile time.
- Preserving **backward compatibility** with older versions of Java.

Type Inference

- Type inference is when the compiler can look at the type of a method argument to infer a generic type.

Generics and Primitive Data Types

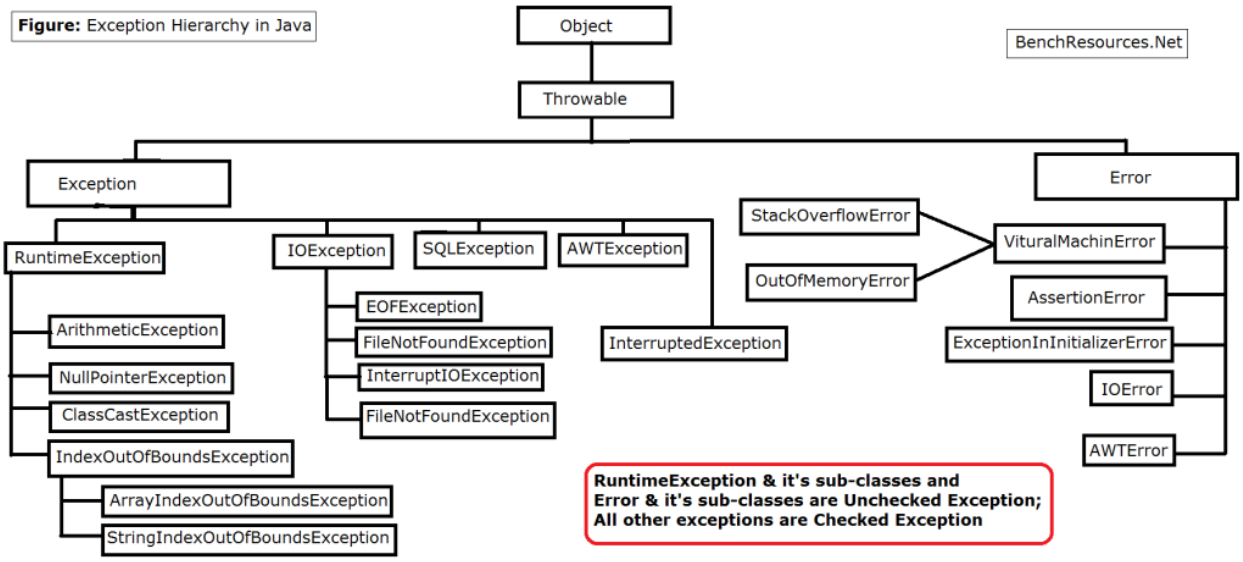
- **One restriction of generics in Java is that the type parameter cannot be a primitive type.**
- **But primitive type arrays can be passed to the type parameter because arrays are reference types.**

ArrayList<int[]> a = new ArrayList<>();

Advantages:

- **Type safety:** by using generics we have a compile type check which prevents ClassCastExceptions and removes the need for casting.
- **Code Reusability**

IV Exceptions



What is an Exception?

An excepton is a rare occurrence that occurs during the execution of a programme and causes the normal flow of the program's instructions to be disrupted.

- Exception are due to ***programmatic logic***
- And it is ***recoverable***

- Exception are categorized into ***checked exception and unchecked exception***
- **Example:** RuntimeException, SQLException, IOException, FileNotFoundException, ArithmeticException, NullPointerException

4.1 Checked Exception:

For the smooth execution of the program, the compiler checks whether the programmer handled a particular exception or not which might occur during the runtime of the program. This type of exception is called a Checked exception.

4.2 Unchecked Exception:

The exception which is not checked by the compiler is called an Unchecked exception.

4.3 Error:

- Error are due to ***lack of system resources***
- And it is ***non-recoverable***
- All error fall into ***unchecked exception*** category, as it is raised due to lack of system resources at runtime
- It is ***out of programming scope*** as such type of error can't predicted, may be well planned care can be taken to avoid these kind of Error
- **Example:** VirtualMachineError, AssertionError, ExceptionInInitializerError, StackOverflowError, OutOfMemoryError, LinkageError, InstantiationError

4.4 try catch:

```
try {
    // program code that
    // could raise or throw exception
}catch(ExceptionType var) {
    // handle exception here
    // provide alternative solution or way
}
```

4.5 Finally block:

The finally block code is always executed regardless if an exception has occurred or not, exception handled or not. This block especially can be used to maintain all cleanup code, such as close DB connection etc.

V Collection FrameWork

VI Object Cloning

6.1 Shallow Clone

6.2 Deep Clone