Advanced Proramming with Java

Prativa Nyaupane

Recap

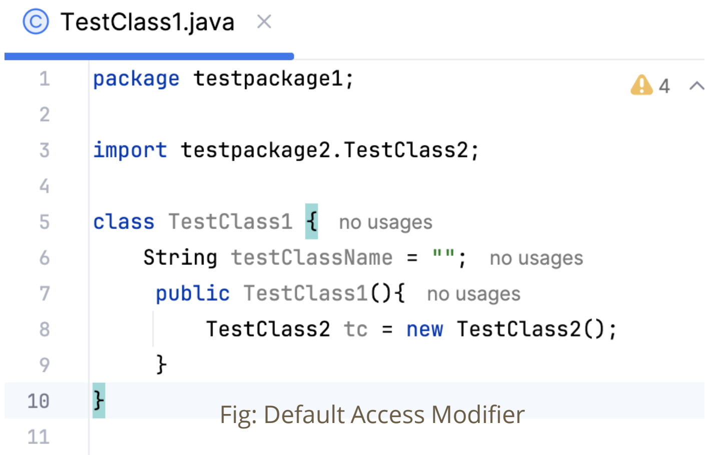
- Variables and Data Types
- Conditional Statements

Today’s Objectives

- Access Modifiers
- Exception Handling
- Java Collections
- Review of object oriented principles
- Super class, sub class, inheritance and member access

Access Modifiers

- Java provides access modifiers to control the visibility and accessibility of classes, variables, methods and fields.
- Provides security, accessibility to the user depending upon the access modifier used with the element.
- Four types of access modifiers are:
    1. Default:
        - When no access modifier is specified, then the class, data members or methods have default constructor.
        - It restricts access to the same package
    1. Private:
        - It is specified using the keyword "private".
        - Any other class of the same package will not be able to access these members



TestClass1.java

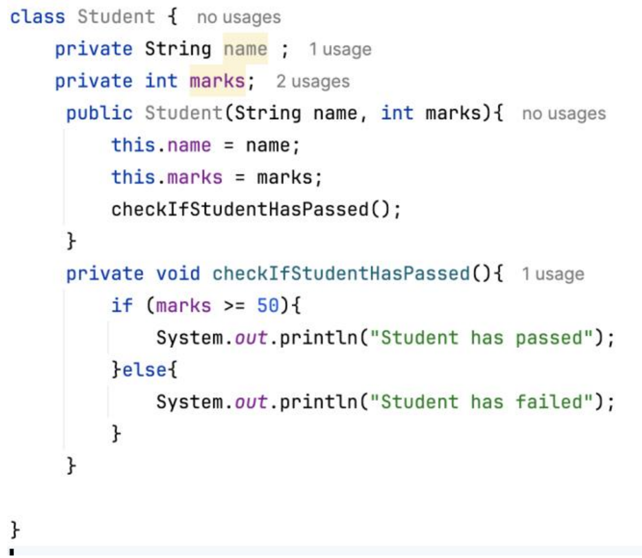
```java
package testpackage1;

import testpackage2.TestClass2;

class TestClass1 {  no usages
    String testClassName = "";  no usages
     public TestClass1(){  no usages
        TestClass2 tc = new TestClass2();
     }
}

```

Fig: Default Access Modifier

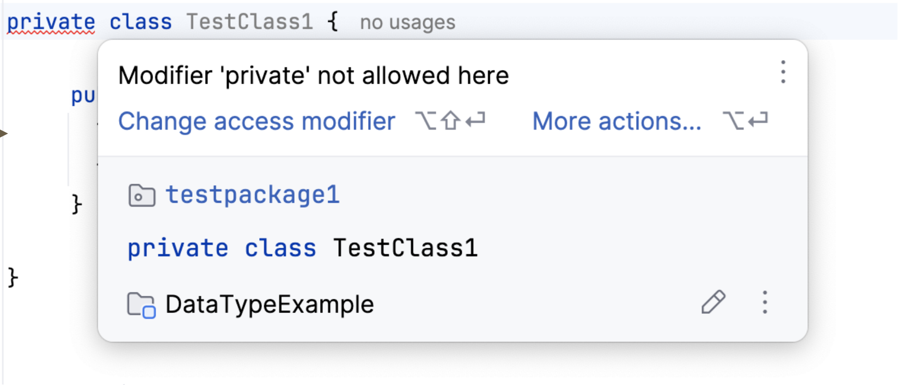


```java
class Student {  no usages
    private String name ;  1 usage
    private int marks;  2 usages
     public Student(String name, int marks){  no usages
        this.name = name;
        this.marks = marks;
        checkIfStudentHasPassed();
     }
     private void checkIfStudentHasPassed(){  1 usage
        if (marks >= 50){
            System.out.println("Student has passed");
        }else{
            System.out.println("Student has failed");
        }
     }

}
```

2. Private contd…

- Top level classes or interfaces cannot be declared as private because data members or methods declared using "private" is only accessible within the enclosing class.



```
private class TestClass1 {  no usages

    pu

    }

}
```

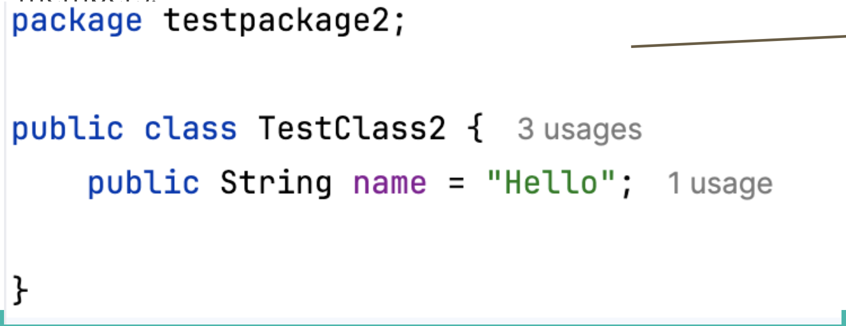
Modifier 'private' not allowed here
Change access modifier ⌥⇧↵   More actions… ⌥↵

```
testpackage1
private class TestClass1
```

DataTypeExample

3. Public:

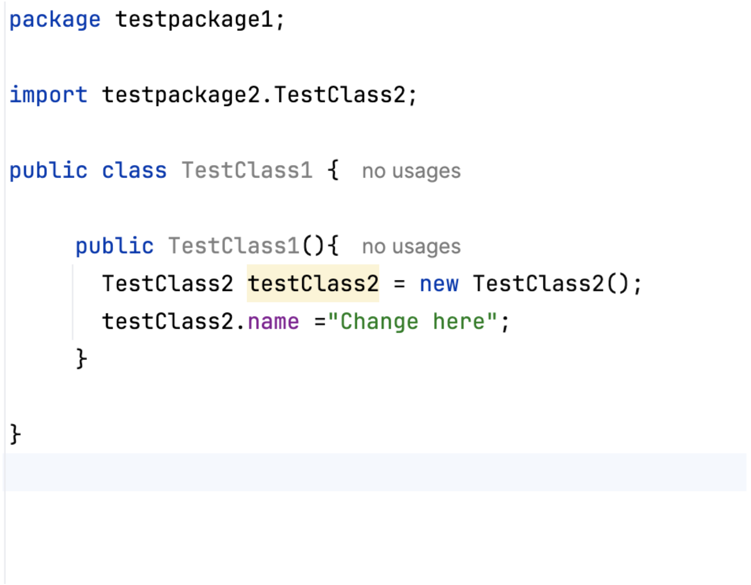
- Specified using "public" keyword.
- Has the widest scope. Classes, methods or data that are declared public are accessible from everywhere in the program.
- There is no restriction on the scope of public data members.



```
package testpackage2;

public class TestClass2 {  3 usages
    public String name = "Hello";  1 usage

}
```



```
package testpackage1;

import testpackage2.TestClass2;

public class TestClass1 {  no usages

    public TestClass1(){  no usages
        TestClass2 testClass2 = new TestClass2();
        testClass2.name ="Change here";
    }

}
```

Access Modifiers Contd..

4. Protected:

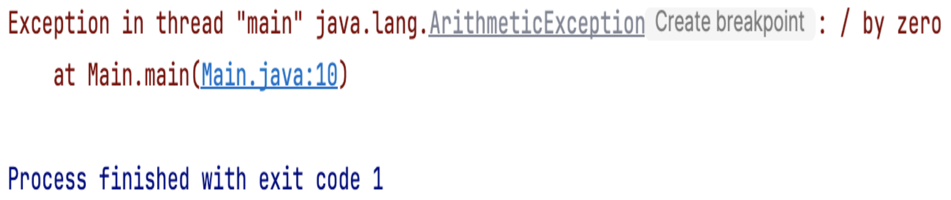
- Specified using the keyword protected.
- The methods or data members declared as protected are accessible only within the same package or subclasses in different packages

Exception Handling

- Exception occurs when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do.
- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
- In Java, an exception is an event that disrupts the normal flow of the program.
- It is an object which is thrown at runtime.

```
public class Main {
   public static void main(String[]
            args) {
        System.out.println(1 / 0);

   }}
```

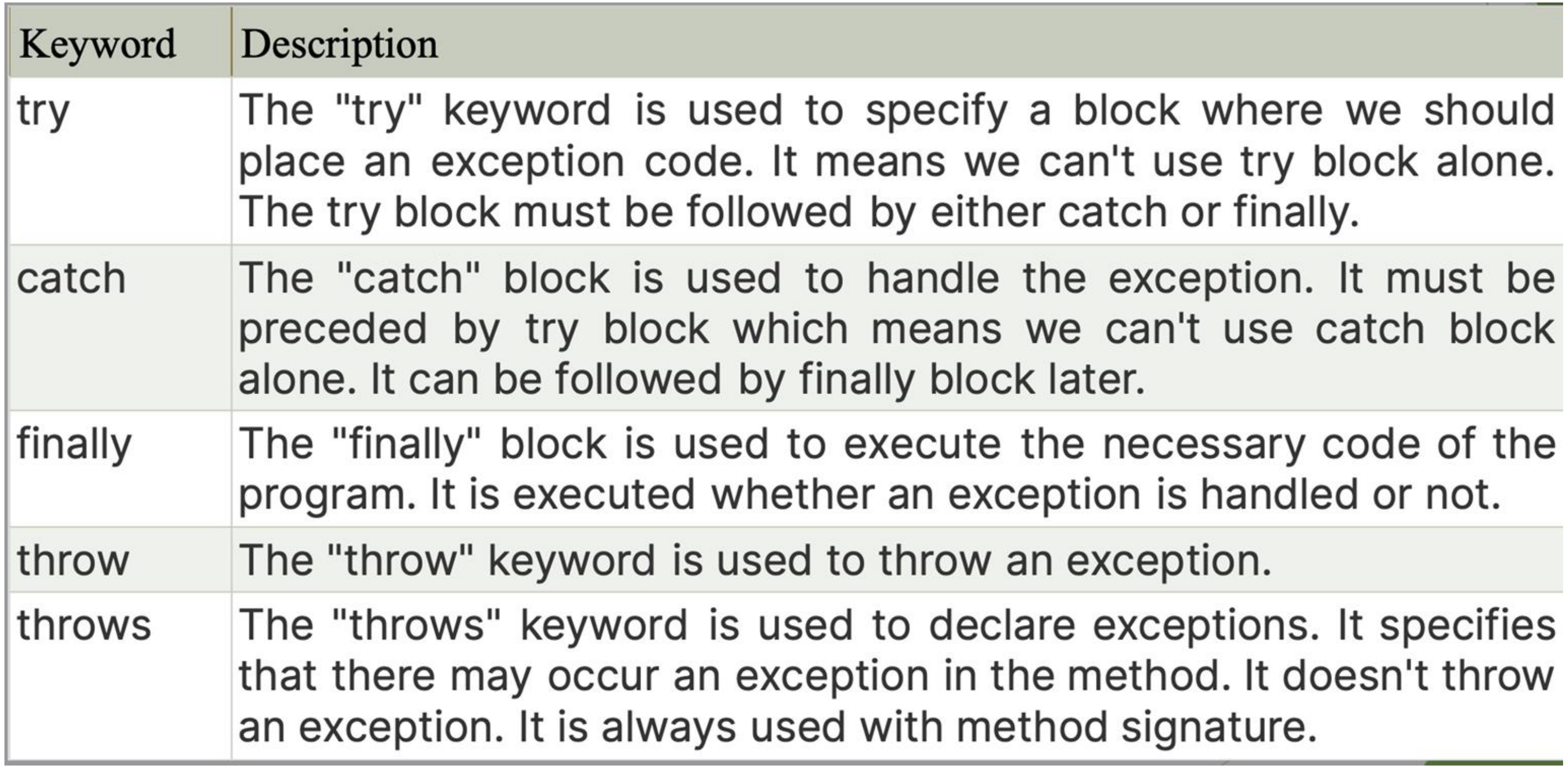


```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.main(Main.java:10)


Process finished with exit code 1
```

- The core advantage of exception handling is to maintain the normal flow of the application

Exception Handling (contd...)



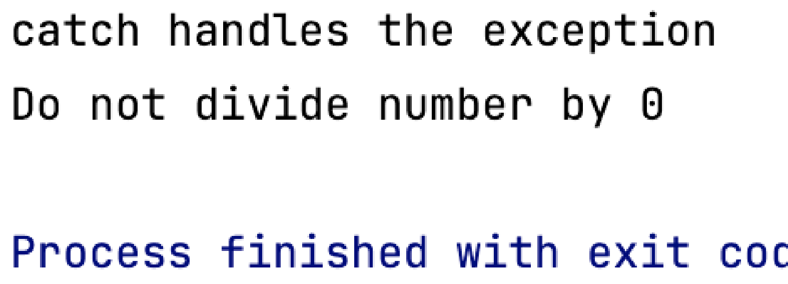
Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Exception Handling Contd..

```java
public class Main {

   public static void main(String[] args) {
       try {
           System.out.println(1 / 0);
       }catch (ArithmeticException arithmeticException){
           System.out.println("catch handles the
exception");
       }finally {
           System.out.println("Do not divide
number by 0");
       }
   }

}
```



```
catch handles the exception
Do not divide number by 0

Process finished with exit co
```

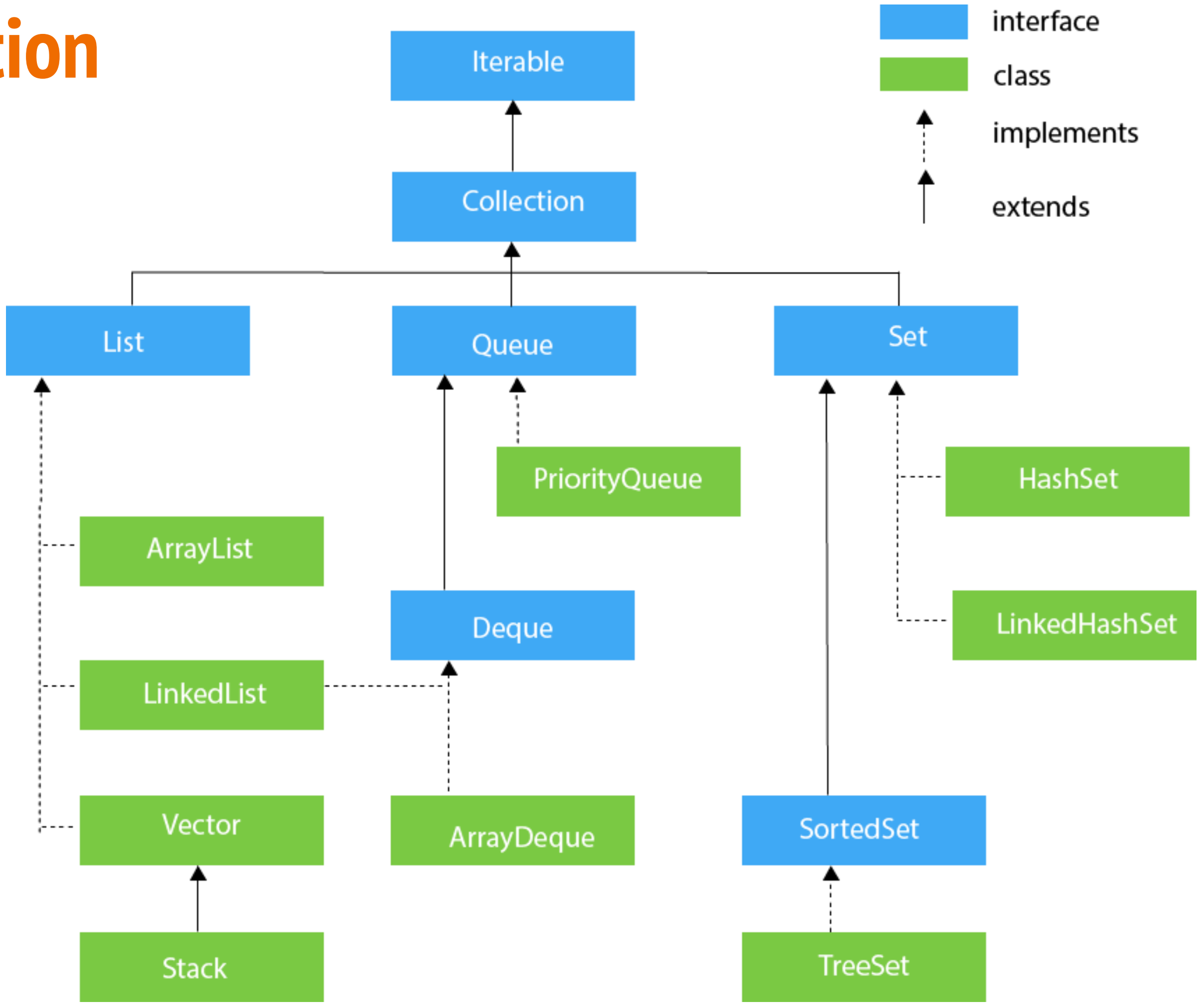
Collections

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

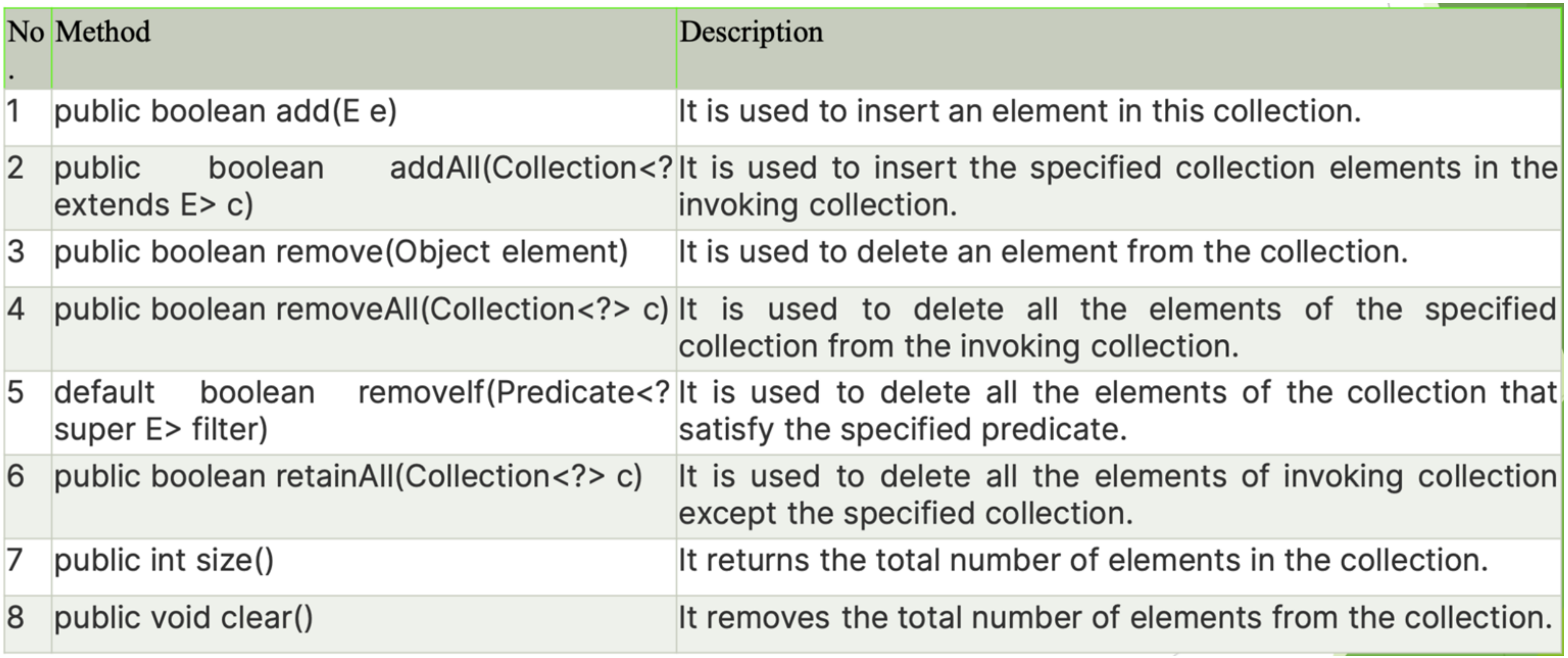
What is a framework in Java?

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

Hierarchy of Collection Framework



Methods of Collection Interface



No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.

Object Oriented Principles

Review of Object Oriented Principles

- OOP is a programming paradigm, where entities, which can represent real-world objects, such as cars or animals are represented using attributes(properties) and behaviors(actions or functions).
- OO Principles are the fundamental concepts that guide the design and implementation of object-oriented systems.
- These principles help in creating modular, maintainable and reusable code.

The main object oriented principles are:

1. Objects
    a. In java, everything is encapsulated within classes and objects. Any object, that comes from that respective class, are essentially reusable software components.
    b. There are data objects, time objects, audio objects, video objects and so on.
    c. Almost any noun can be reasonably represented as a software object in terms of attributes and behaviours.
    d. For example: A car has a brand and a model and it can go from one place to another.
2. Classes and methods
    a. Classes define the blueprint for creating objects, which are instances of classes.
    b. In Java, class is a unit to house the set of methods and attributes.
    c. The method contains the program statements that actually performs its tasks.
    d. For example, a bank account may contain one method to deposit money to an account and another method to withdraw money and a third method to inquire what the account's current balance is.
    e. A class is instatianted and an object is created. The object,in turn calls the methods.
    f. Classes are reusable components.

3. Encapsulation:

a. Classes(and their objects) encapsulate i.e encase their attributes and methods.
b. A class's (amd its objects) attributes and methods are closely related.
c. Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented - implementation details are hidden within the objects themselves.

4. Inheritance:

a. A new class objects can be created conveniently by inheritance - the new class(called the subclass) starts with the characteristics of an existing class(called the superclass) possibly customizing them and adding unique characteristics of its own.
b. This promotes code reuse and facilitates hierarchical classification of objects.

5. Polymorphism:

a. Polymorphism is a concept by which we can perform a single action in different ways.
b. Polymorphism in Java can be performed through method overriding and method overloading.
c. If a class has multiple methods having same name but different parameters, it is method overloading.
d. If a subclass has same method as parent, it is called method overriding.

Method Overloading

```
public class Student {
   private String name;
   private int rollNumber;
   public Student() {
   }
   public void setStudentAtrributes(String name) {
       this.name = name;
   }

   public void setStudentAttributes(String name,
int rollNumber){
       this.name = name;
       this.rollNumber = rollNumber;
   }

}
public class Main {
   public static void main(String[] args) {

       //Student student = new Student();
       Student student = new Student();
       student.setStudentAtrributes("Ram Poudel");
       student.setStudentAttributes("Ram
Paudel",1);


   }
}
```

Method Overriding

```
public class AnimalPolymorphismEx {

   public void greeting() {
       System.out.println("The animal greets you.");
   }


}
public class DogPolyMorphismExample extends
AnimalPolymorphismEx {

   public void greeting() {
       System.out.println("The dog barks.");
   }

}
public class PolymorphismExample {

   public static void main(String[] args) {

       AnimalPolymorphismEx animalPolymorphismEx1 = new
AnimalPolymorphismEx();  // Animal object
       animalPolymorphismEx1.greeting(); // prints "The
animal greets
       AnimalPolymorphismEx dog = new
DogPolyMorphismExample();  // Dog object
       dog.greeting(); // prints "The dog barks

   }
}
```