getClass()

- It returns the runtime class of an object.

```
package getclassexample;

public class Main {
    public static void main(String[] args) {
        String str = "Flynaut";
        Integer num = 10;
        Test testObj = new Test();

        System.out.println("Class of str: "+
str.getClass().getName());
        System.out.println("Class of num: "+
num.getClass().getName());
        System.out.println("Class of testObj: "+
testObj.getClass());
    }
}
```

EQUALS METHOD:

```
// emp1.equals(emp2)
    public boolean equals(Object obj) {
        if (this == obj) return true; // same reference check
        if (obj == null || getClass() != obj.getClass())
return false;

        Employee employee = (Employee) obj;
        return id == employee.id &&
name.equals(employee.name);
    }
```

emp1.equals(emp2);

1. if (this == obj) return true;

   this: refers to the current object which calls the equals() method

   == : Checks if the two references (memory addresses) point to the same object.

2. if (obj == null || getClass() != obj.getClass()) return false;

- obj == null -> checks if the passed object is null
- If obj is null, return false because a null object cannot be equal to a valid object.
- getClass() != obj.getClass() = Checks if the classes of two objects are different
  getClass()-> returns the runtime class of current object(emp1)
  obj.getClass()-> returns the runtime class of the passed object
  If classes are not same-> return false

3. Employee employee = (Employee) obj;

(Employee) obj : this is a downcast which converts obj from Object to Employee so that properties of Employee can be compared

4. return id == employee.id && name.equals(employee.name);

- id == employee.id -> Compares the id values of both objects. If they are equal it returns true, otherwise it will return false.
- name.equals(employee.name)- comparing name values of both the objects

- name is a String, so .equals is used to check the value equality.

- && -> LOGICAL AND operator
  Both the conditions must be true to return the true
  T T -> T
  If either condition is false, the method returns false.

  Without downcasting
  Obj.id -> Compilation Error

Comparable & Comparator

These are interfaces which are used for sorting objects

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Comparable.html

Comparator (Java SE 21 & JDK 21)

- Comparable :
  It is used to define natural ordering of objects.
  Natural Ordering:

- Strings -> alphabetical order(A-Z,a-z)
- Numbers -> ascending order(1,2,3,4,……)

- Comparator:
  It is used to define the custom ordering.

1. Comparable:
   It is an interface which is a part of java.lang package
   It has a single method called compareTo, which is used for defining natural order of objects.

SYNTAX:

```
Public class Employee implements
Comparable<Employee>{

  @Override
  Public int compareTo(Employee obj){
//Comparison logic
      }
}
```

```
package comparableExample;

public class Employee implements
Comparable<comparableExample.Employee>{
    private String name;
    private int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
```

```java
                ", age=" + age +
                '}';
    }


    @Override
    public int compareTo(Employee o) {
        return this.age - o.age;    //Natural sorting with age
    }
}
```

```java
package comparableExample;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Krishna",30));
        employees.add(new Employee("Gopal",20));
        employees.add(new Employee("Govind",50));

        Collections.sort(employees);

        for (Employee emp: employees){
            System.out.println(emp);
        }

    }
}
```

```java
@Override
    public int compareTo(Employee o) {
        return this.age - o.age;    //Natural sorting with age
    }
```

The compareTo() method compares the current object(this) with the specified object(o).

- 0 -> if both objects are equal
- + -> the current object is greater

- - -> the current object is smaller

1. If this.age is less than o.age, the result will be negative(indicating this comes before o)
2. If this.age is greater than o.age, the result will be positive(indicating this comes after the o)
3. If this.age equals o.age,the result will be 0(both are equal)