# Object equals() method:

- We can use `equals()` method to check the equality of to objects. `obj1.equals(obj2)`
- If our class doesn't `equals()` method then object class `equals()` method will be executed.

```
class Student {
    String name;
    int rollno;

    Student (String name, int rollno) {
        this.name = name;
        this.rollno = rollno;
    }

    public static void main(String[] args) {
        Student s1 = new Student("Ahmed", 101);
        Student s2 = new Student("John", 102);
        Student s3 = new Student("Ahmed", 101);
        Student s4 = s1;

        System.out.println(s1.equals(s2));  => false
        System.out.println(s1.equals(s3));  => false
        System.out.println(s1.equals(s4));  => true
    }
}
```

- In the previous example Object class `equals()` method got executed which is meant for reference comparison (address comparison), that is if two references pointing to the same object then only that `equals()` method returns *true*.
- Based on our requirements we can override `equals()` method for content comparison.

# Overriding equals() method:

- While *overriding* equals() method for content comparison we have to take the following things:
- What's the meaning of equality (wether we have to check only name, or only rollno or both)
- If we are passing different type of object, our `equals()` method should not raise `ClassCastException`, which means we have to handle `ClassCastException` to return `false`
- If we are passing `null` agument then our `equals()` method should not raise `NullPointerExecption`, which means we have to handle `NullPointerException` to return `false`.
- The following is a ***proper*** way of *overriding* `equals()` method for `Student` class content comparison.

```
  class Student {
      String name;
      int rollno;

      Student (String name, int rollno) {
          this.name = name;
          this.rollno = rollno;
      }

      public boolean equals(Object obj) {
          try {
              String name1 = this.name;
              int rollno1 = this.rollno;

              Student s = (Student)obj;   => runtime exception:
ClassCastException

              String name2 = s.name;      => runtime exception:
NullPointerException
              int rollno2 = s.rollno;     => runtime exception:
NullPointerException

              if (name1.equals(name2) && rollno1 == rollno2) {
                  return true;
              }
              else {
                  return false;
              }
          }
          catch(ClassCastException e) {
              return false;
          }
          catch(NullPointerException e) {
              return false;
          }
      }

      public static void main(String[] args) {
          Student s1 = new Student("Ahmed", 101);
          Student s2 = new Student("John", 102);
          Student s3 = new Student("Ahmed", 101);
          Student s4 = s1;

          System.out.println(s1.equals(s2));      => false
          System.out.println(s1.equals(s3));      => true
          System.out.println(s1.equals(s4));      => true
          System.out.println(s1.equals("Ahmed")); => false (if
ClassCastException is not handled, it would be raised at runtime)
[String is a diffrent type of object]
          System.out.println(s1.equals(null));    => false (if
NullPointerException is not handled, it would be raised at runtime)
      }
  }
```

`s1.equals(s2)` => calling the equals() method on s1, thus we have two objects. `s1` <=> `this`

## Simplifying equals() method:

- ```
  public void m1() {
      System.out.println(x);
      System.out.println(this.x);
  }
  ```

  Inside an *instance* method accessing any variable directly means we are accessing the current *object* variable.
  This is why we can simplify the `equals()` method like this:

- ```
  public boolean equals(Object obj) {
      try {
          Student s = (Student)obj;
          if (name.equals(s.name) && rollno == s.rollno) {
              return true;
          }
          else {
              return false;
          }
      }
      catch(ClassCastException e) {
          return false;
      }
      catch(NullPointerException e) {
          return false;
      }
  }
  ```

  A more simplified version of `equals()` method by removing the `try catch` block by using `instanceof` operator:

- ```
  public boolean equals(Object obj) {
      if (obj instanceof Student) {
          Student s = (Student)obj;
          if (name.equals(s.rollno) && rollno == s.rollno) {
              return true;
          }
          else {
              return false;
          }
      }
      return false;
  }
  ```
  ```

A more efficient version of `equals()` method:

- ```
  public boolean equals(Object obj) {
      if (this == obj) {
          return true;
      }

      if (obj instanceof Student) {
          if (name.equals(s.rollno) && rollno == s.rollno) {
              return true;
          }
          else {
              return false;
          }
      }
      return false;
  }
  ```

According to `this == obj` if both reference pointing to the same object, then without performing any comparison `equals()` method returns `true` directly.

**Note:**

```
String s1 = new String("Ahmed");
String s2 = new String("Ahmed");

System.out.println(s1 == s2);        => false
System.out.println(s1.equals(s2));   => true
```

`equals()` method in `String` class is overriden for content comparison.

```
String s1 = new StringBuffer("Ahmed");
String s2 = new StringBuffer("Ahmed");

System.out.println(s1 == s2);        => false
System.out.println(s1.equals(s2));   => false
```

`equals()` method in `StringBuffer` class is not overriden for content comparison, thus `Object` class `equals()` method will be called and perfoms reference comparison.
```