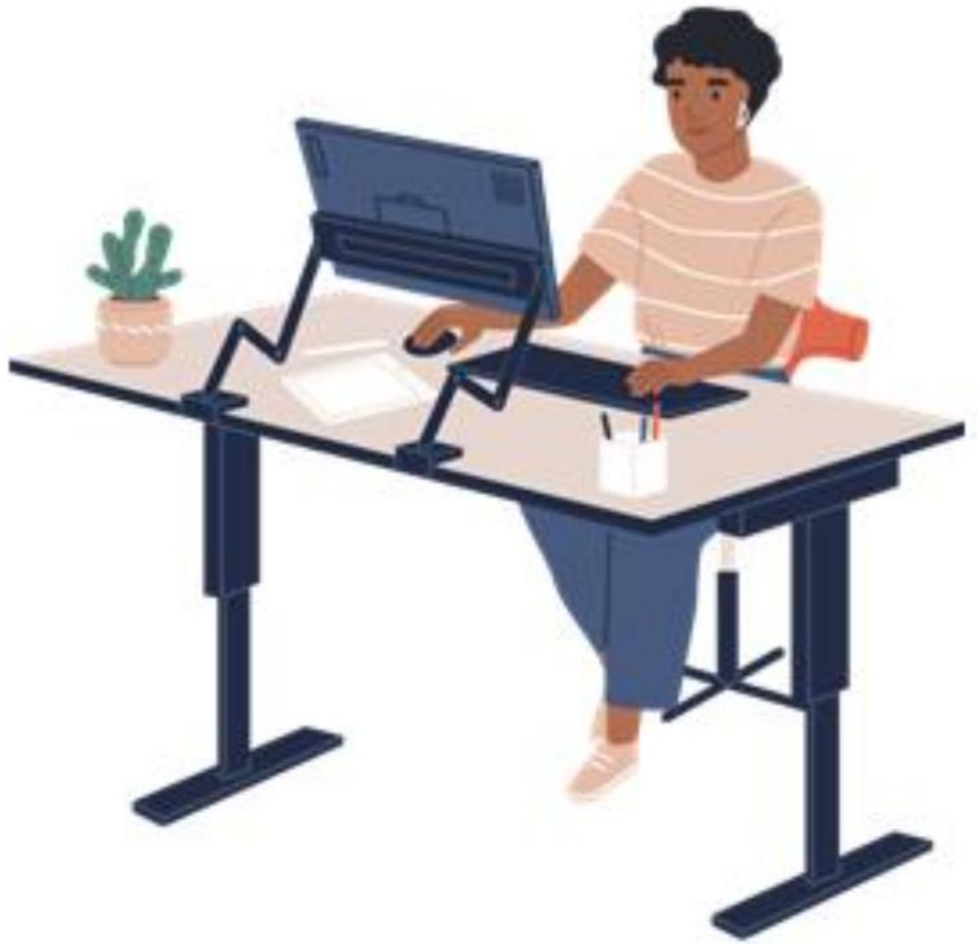
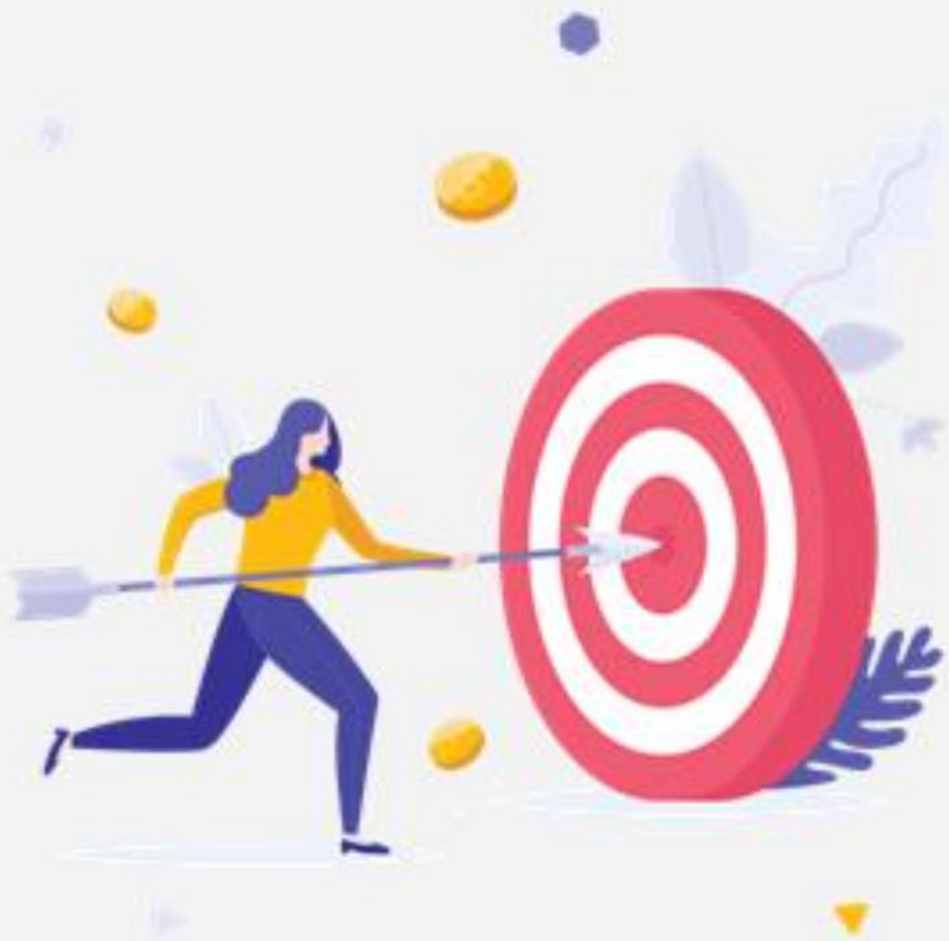
Learning Consolidation

Implement Inheritance





Learning Objectives

- Explore relationship between objects
- Explore the methods of the Object class

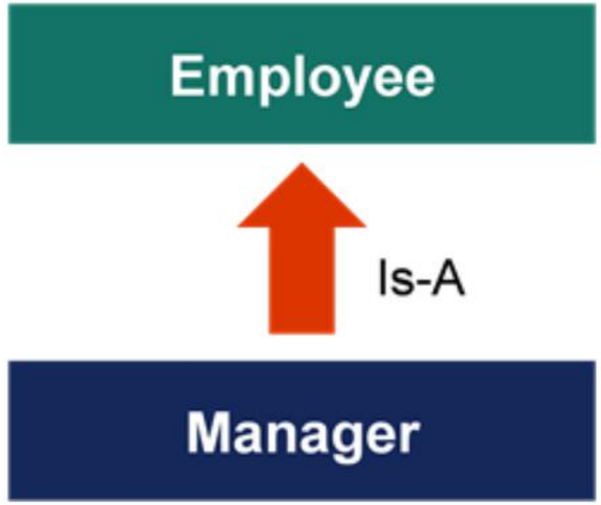
Relationship Between Objects – Composition

- Java objects can be related to one another just as in the real world.
- When an object contains another object, it's is called as composition.
- A Java object Employee can have an address.

- The **has-a** relationship is referred to as a Composition relationship between objects.

Relationship Between Objects – Inheritance

- Java objects can have a hierarchical relationship to one another just like real-world objects.
- In an organization, a Manager or Programmer can be the Employee of an organization.
- A Manager is-a Employee.
- The **is-a** relationship between objects is called as Inheritance relationship.
- In this case, we can say that the Manager class can be derived from the Employee class.



The Object Class

- The `Object` class of the `java.lang` package is the parent class for all the classes in Java.
- It is the topmost class in the class hierarchy.
- All classes implicitly inherit the Object class.
- The following methods of the Object class that are used extensively:
  - `protected Object clone() throws CloneNotSupportedException`
  - `protected void finalize() throws Throwable`
  - `public final Class getClass()`
  - `public String toString()`
  - `public boolean equals(Object obj)`
  - `public int hashCode()`

The `clone()` Method

- The `clone()` method helps us to create or make a copy of an existing object. If we want to clone an object of a class, the rules to be followed are as below:
  - The class must override the `clone()` method of the Object class.
  - Every class that is implementing `clone()` must call `super.clone()` to get the clone's object reference.
  - The class should implement the interface `java.lang.Cloneable`, whose object clone you want to create, otherwise it will throw `CloneNotSupportedException` whenever a clone method is called on that class's object.
- There are two types of cloning:
  - Deep cloning
  - Shallow cloning

The `equals()` Method

- The `equals(Object obj)` is a method of an `Object` class that can be used to do the comparison between the given objects.
- It is advised to override `equals(Object obj)` method to get your own equality condition on the objects.

The `equals()` Method

- The `Money` class shown gives implementation of `equals` method in the `Money` class.

```
class Money
{  int amount;
    String currencyCode;
    public Money(int amount, String currencyCode) {
        this.amount = amount;
        this.currencyCode = currencyCode;
    }
    @Override
    public boolean equals(Object o) {
        if (o == this)
            return true;
        if (!(o instanceof Money))
            return false;
        Money other = (Money)o;
        boolean currencyCodeEquals = (this.currencyCode == null && other.currencyCode == null)
                || (this.currencyCode != null && this.currencyCode.equals(other.currencyCode));
        return this.amount == other.amount && currencyCodeEquals;
    }
}
public class MoneyImpl {
    public static void main(String[] args) {
        Money income = new Money( amount: 55,  currencyCode: "USD");
        Money expenses = new Money( amount: 55,  currencyCode: "USD");
        boolean balanced = income.equals(expenses);
        System.out.println("Income is equal to Expenses " + balanced);
    }
}
```

The `toString()` Method

- The `toString` method is used to return a string representation of an object.
- If any object is printed as shown below, the `toString()` method is internally invoked by the java compiler.

```
Author author = new Author();
System.out.println(author);
```

- The `toString` method can be overridden to give our own implementation in the `Author` class as below,

```
@Override
public String toString() {
    return "Author{" +
            "authorName='" + authorName + '\'' +
            ", authorPenName='" + authorPenName + '\'' +
            '}';
}
```

- **Note**: More on overriding will be discussed in later Sprints.