Inheritance

What is Inheritance?

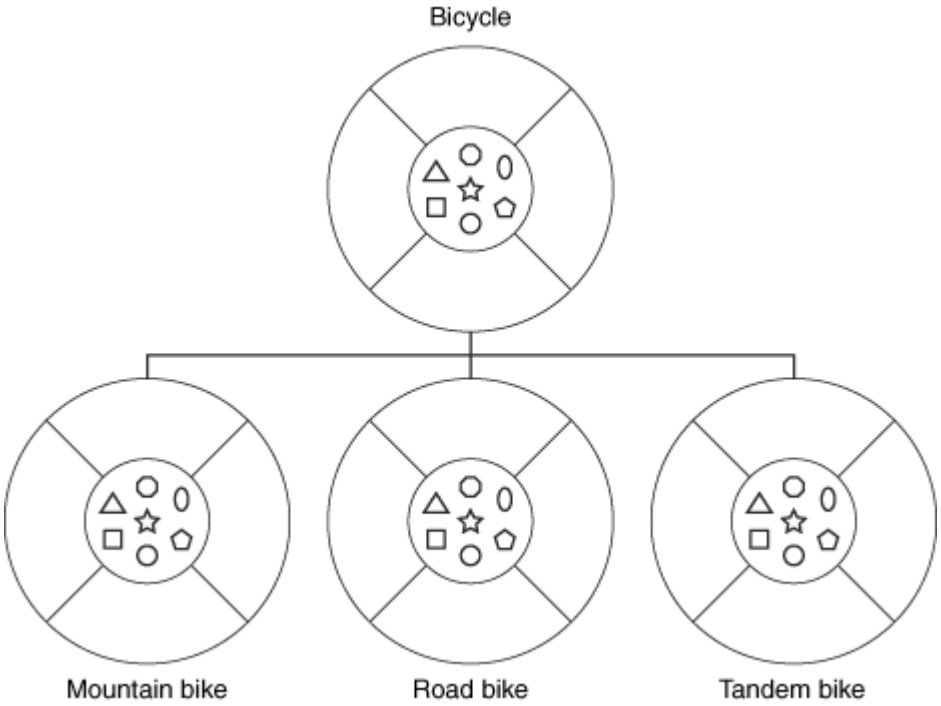
A process by which a class can extend another class, inheriting all its

- variables,
- methods,
- while redefining (overriding) some of them,
- and/or adding its own.

Thus inheritance promotes code re-use. If a class “B” needs to define some behavior and a lot of that behavior is pre-defined in any other class “A” then instead of defining the entire behavior again, the class “B” should inherit from that class “A”.

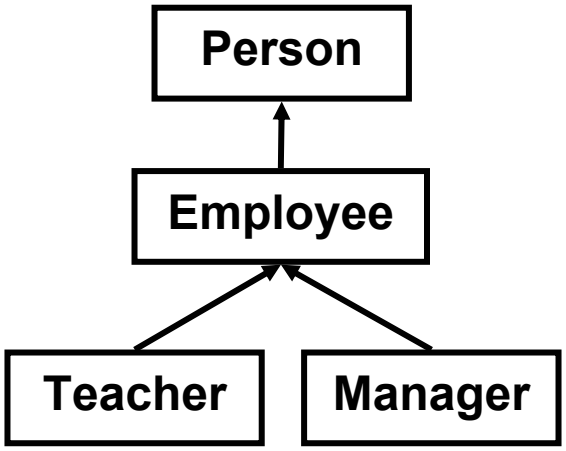
An Example

Inheritance allows classes to be defined in terms of other classes. For example, mountain bikes, road bikes, and tandems are all kinds of bicycles. In object-oriented terminology, mountain bikes, road bikes, and tandems are all **subclasses** of the bicycle class. Similarly, the bicycle class is the **superclass** of mountain bikes, road bikes, and tandems. This relationship is shown in the following figure.



Tree Hierarchy

Inheritance leads to hierarchy of classes. The classes arranged in tree hierarchy increases the understanding. For example consider the following tree hierarchy



Inheritance Vocabulary (Buzzwords)

- **Superclass**
  - A superclass is the class above in the hierarchy. For example Person is a direct superclass of Employee and indirect superclass of Teacher or manager.
  - Sometimes also referred as *parent* or *base* class.
  - A superclass has fewer properties (as compared to subclass).
  - A supeclass is more *general* For example; Employee class is a general class as compare to Teacher as it (Employee) can represent teacher, manager, secretary, and clerical staff and so on.
- **Subclass**
  - A subclass is the class below (of super class) in the hierarchy. For example, Employee is a direct subclass of Person; also teacher is indirect subclass of Person but direct subclass of Employee.
  - Sometimes also refer as *child* or *derived* class.
  - A subclass generally has more properties (as compared to superclass) because by the rule of inheritance, a subclass inherits properties of superclass as well as defining/adding its (subclass) own. Tandem bicycles have two seats and

two sets of handle bars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.

- A supeclass is more *specific* For example; Teacher class is a specific class as compare to Employee as it can only represent Teacher's category (Professors, Lecturers and Instructors etc) as compared to employee (general) class which can represent Teacher as well as Managers

- **Sibling**

  - Two children (subclasses) of the same parent class are called siblings. In the above example, Teacher and Manager are siblings.

"Is a/an" relationship

Subclass *is a* Superclass or an instance of a subclass *is an* instance of superclass since a subclass has all the properties of a superclass and some additional properties as well.

Considering the above class hierarchy, we can say that Teacher *is an* Employee and rightly so as teacher is an employee from the university perspective. Also Manger *is an* Employee & Employee *is a* Person.

Since Employee is parent of Teacher and Person is a parent of Employee, so by following the inheritance rule, Teacher *is an* Employee *is a* Person. So, Person is indirect parent of Teacher. By summing up the above discussion, we can conclude that Teacher *is a* Person.

Inheritance Warning

Inheritance is the core part of OOP. Inheritance is a clever and appealing concept. However using it properly needs clarity of concept. A common error made by beginner OOP programmers is that they do not make use of inheritance. Let's make our concepts clear from the following example

Horse/Zebra Example

- With inheritance, we define classes in terms of other classes. This can be a great shortcut if the classes are similar

.

- Suppose you have the hierarchy of all the animal classes, except the zebra class.
- Now, you want to make a zebra class. What to do?
  - Option 1 - define the zebra class from scratch (Wrong)
  - Option 2 - Locate the horse class. Make Zebra class a subclass of Horse class
- Benefits – Zebra inherits 90% of its behavior so NO coding requirement + time save.
- In the Zebra class, define the few things that are features of Zebras but not horses.
- What would you do if horses have some features that Zebras don't have
  - You'll see if the parent class of horses can do the job. Then you will inherit from the parent class of horse and not horse itself.

Points to Remember

- Java doesn't support multiple inheritances, it only support single inheritance. It means that a class can only extend from one class at one time.
- All java classes inherit from Object class (more on it later).
- Things declared "public" in superclass are available/accessible to subclass and all other classes as well.
- Things declared "protected" in superclass are available/accessible to subclass and not to other classes.
- Things declared "private" are not directly accessible to subclass or any other class.

- Whenever you construct the object of subclass, subclass constructor would be called as well as superclass constructor would also be called.

  - If you don't specify which superclass constructor to be called, the java will automatically call the default constructor of the superclass. (Example of these are coming up shortly).

Comparison with C++

See handout on "Similarities and Differences between C++ & Java".

Constructors are Not Inherited

- Although a subclass inherits methods and variables from the parent class, it does not inherit constructors.

- If you don't write any constructor (no argument or parameterize) of subclass than java will write default constructor for you.

Method Overriding

- The process of redefining a method, that is already defined in the superclass with same signature (name + parameters) and return type, in the subclass is called method overriding.

- Overriding allows the subclasses to modify behavior (method) as needed.

- Suppose that area method is defined in the Shape class and Circle class is a subclass of Shape class. So, area method through inheritance is available to Circle class but it is useless for Circle class since the area calculating methodology of circle (πr^2) is different of Shape. As a result, Circle class has to redefine area method (overriding) to achieve the desired behavior.

Inheritance using Java

Keyword *extends*

Keyword *extends* is used for defining inheritance between super and sub class as the subclass is extending the behavior and functionality of superclass.

```
class Teacher extends Employee {

    ……….

}
```

Keyword *super*

The keyword super is used for two purposes:

- To call the superclass constructor from the subclass constructor
- To call the superclasss method from subclass that is overridden in the subclass

Both usage of the super keyword is shown in the coming up code example.

Code Example

Employee.java

The Employee class has name and id as instance variables. Two constructors, default & parameterized are also provided. Other methods include setter/getters, display (capable of printing employee object on console) and toString method (capable of converting employee object into string) is overridden here.

Note: The toString() method is actually defined in the Object class, and every class in java inherits from Object class whether *extends* kewords is specified or not

```
class Employee{

  protected int id;
  protected String name;

  //parameterized constructor
  public Employee(int id, String name){
     this.id = id;
     this.name = name;
  }
```

```
  //default constructor
  public Employee(){
      this (10, "not set");
  }

  //setters
  public void setId (int id) {
     this.id = id;
  }

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

  //getters
  public int getId () {
     return id;
  }

  public String getName () {
     return name;
  }

  // displaying employee object on console
  public void display(){

     System.out.println("in employee display method");

     System.out.println("Employee id:" + id + " name:" + name);
  }

  //overriding object's class toString method
  public String toString() {

     System.out.println("in employee toString method");

     return "id:" + id + "name:" + name;
 }

}//end class
```

Consider the following line of code in the default constructor of the Employee's class.

```
this (10, "not set");
```

As you already know, *this* is the keyword and here it is used to call the Employee class parameterize constructor. Hence, Keyword *this* can be used to call the other constructor of the same class. For example, to call the default constructor, write following code:

```
this ( );
```

Teacher.java

The Teacher class extends from Employee class. The teacher class has an additional attribute i.e. qualification. Two constructors, default & parameterized are also defined. getter/setter is also defined for qualification. Display & toString methods are overridden in this class.

```
class Teacher extends Employee{

  private String qual;

  //default constructor
  public Teacher () {
     //implicit call to superclass default construct
     qual = "";
  }

  //parameterized constructor
  public Teacher(int i, String n, String q){

     super(i,n); //call to superclass const must be first line
     qual = q;
  }

  //setter
  public void setQual (String qual){
     this.qual = qual;
  }

  //getter
  public String getQual(){
     return qual;
  }

  //overriding dispaly method of Employee class
  public void display(){

     System.out.println("in teacher's display method");

     super.display(); //call to superclass display method

     System.out.println("Teacher qualification:" + qual);
  }

  //overriding toString method of Employee class
  public String toString() {

     System.out.println("in teacher's toString method");

     String emp = super.toString();

     return emp +" qualification:" + qual;
  }
}//end class
```

The first usage of keyword *super* is to call superclass's constructors. This has been shown in the parameterize constructor of Teacher class to make call to parameterize constructor of the parent class i.e. Employee class.

Note that we didn't use the *super* keyword in the default constructor of Teacher class but java will implicitly call the default (no argument) constructor of the Employee class.

The second usage of *super* is to call the overridden methods. This has been shown in the display and toString method of the Teacher class.

Points to Remember

- The call to superclass constructor via keyword *super* must be the first line (as used in the parameterized constructor) or otherwise compiler will raise an error.
- Java will make call to superclass default constructor automatically if no one is specified in subclass constructor.
- If the method is not overridden in the subclass than there is no need to use the keyword super before method name.

Test.java

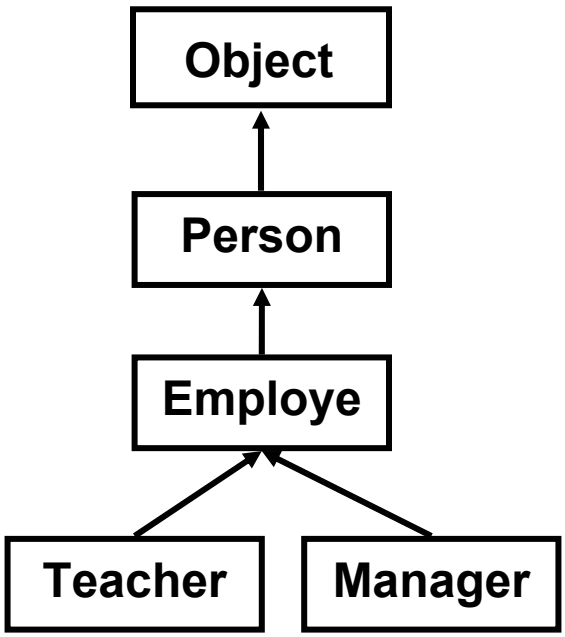
This class acts as a driver class as it contains the main method. Objects of Employee & Teacher class are created inside main and calls are made to display and toString method using these objects.

```
class Test{

  public static void main (String args[]){

     System.out.println("making object of employee");
     Employee e = new Employee(89, "khurram ahmad");

     System.out.println("making object of teacher");
     Teacher t = new Teacher (91, "ali raza", "phd");

     e.display(); //call to Employee class display method
     t.display(); //call to Teacher class display method

     System.out.println("Employee: " +e.toString());

     System.out.println("Teacher: " + t);

  } //end of main
}//end class
```

Object Class

- A class called `Object` is defined in the `java.lang` package of the Java standard class library
- All classes, by default, are derived from the `Object` class i.e. if a class is not explicitly defined to be the child of any superclass class, java makes it the child of the Object class
- Object class contains some useful methods like toString(), equals() (for comparing references) etc.
    - As we didn't specify the superclass of the Employee class. So, java made Object class its superclass. That's why we have overridden toString method in the Employee class.
- The Object class is therefore the ultimate root of all class hierarchies.

The one hierarchy we discussed is modified as one given below assuming that Employee class inherits from Person class. Hence, the Object class will became the superclass of the Person class (ultimate root class).



References:

- Java tutorial: http://java.sun.com/docs/books/tutorial/java/javaOO/
- Stanford university