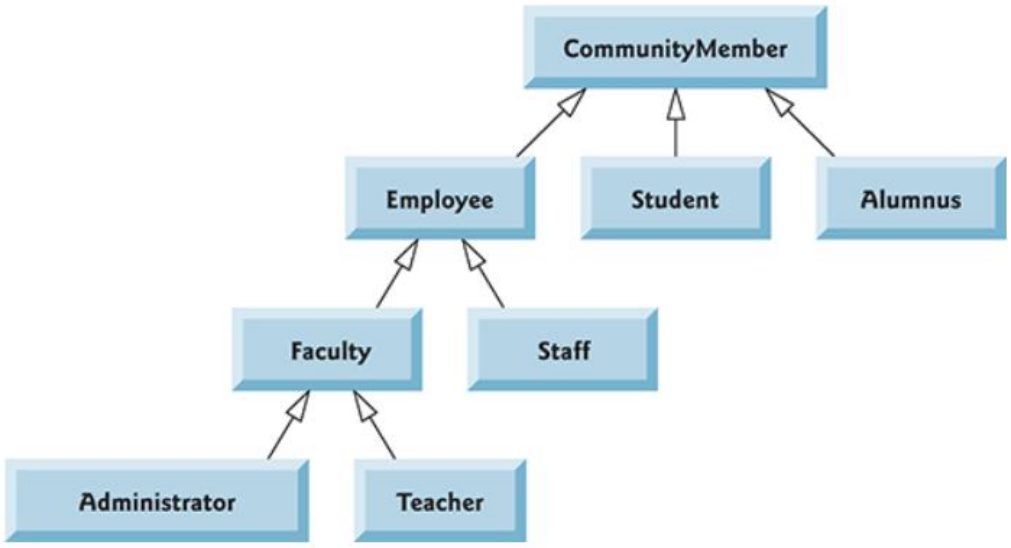
Inheritance

Inheritance – A concept in Object Oriented programming in which a new class is created by acquiring an existing class's members and possibly embellishing them with new or modified capabilities. It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

Much like how one inherits several qualities genetically and materialistically from parents, Java also helps in passing parameters from one class (super or parent class) to another class(sub or child class).

After inheriting the traits (variables and methods) of the super class, a child class can add its own variables to make it more specialized.

Example –



Inheritance hierarchy UML class diagram for university `CommunityMembers`.

All community members have SIN number, first name, last name, address, gender.

Adding to the qualities of community members, Employees can have employeeNum, students can have studentID, alumnus can have alumnusAccessID and so on.

Not only variables, but also methods are inherited from parent class to child classes.

More examples:

Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount

Protected Keyword

Variables and methods that are declared **private** *will not be inherited* as they are not visible outside the class in which they're defined. **Public** members will be inherited but using public for variables will defeat the purpose encapsulation (private variables and public methods) we learnt earlier.

For this purpose, Java has introduced ***protected*** access specifier. Protected variables are accessible only in the super and sub-classes. They will not be visible outside of them. So, ***protected*** keyword comes in very handy when performing inheritance.

Extends keyword

Extends is used to establish super-sub class relationship between two classes.

```java
public class Person {
  protected long SIN;
  protected String fullName;
  public String address;

  public void setSIN(long SIN) {
    this.SIN = SIN;
  }

  public long getSIN() {
    return SIN;
  }

  public void setFullName(String fullName) {
    this.fullName = fullName;
  }

  public String getFullName() {
    return fullName;
  }

 public void print(){
    //both private and public - methods and variables come from parent to c
    System.out.println("SIN is "+SIN);
    System.out.println("FullName is "+fullName);
   System.out.println("Address is "+address);
  }
}
```

```java
public class Student extends Person {

  public void print(){

    System.out.println("SIN is "+SIN);
    System.out.println("Address is "+address
  }
}
```

Here we are able to observe that even without declaring SIN and address in Student, we are still able to use these variables because they have been inherted from Person class and now they have become members of the Student class. This concept is called as **code-reuse ,** that is, we need not re-type the variables and their corresponding getters/setters. Just using the extends keyword does the magic for us.

Note: Only public and protected members will be inherited.

Constructors in Inheritance

Are constructors inherited ? Short answer is NO. But, we can call the parent class constructor from child class constructor

```
public Person(){
  System.out.println("Person constructor");
}
```

```
public Student(){
  /*In the first line of default constructor of any child class, there is a hidden call
o parent class default constructor.

eason : Student is a person. To become
   */

  System.out.println("Student class constructor");
}
```

When you create an object of Student, you will find that the message in Person class's constructor will be printed first and only then message in Student class will be printed.

```
Person constructor 1
Student class constructor
```

This is because, in the first line of constructor of any child class, there is a hidden call to parent class default constructor.

Parameterized constructor

Even in the parameterized constructor of child class, a hidden call to parent class's default constructor is only present.

If you need to invoke the parameterized constructor of the parent class, you must use the ***super*** keyword.

```java
// Parameterized constructor
public Student(long s, String f, String a, String sid, double c) {
  //Even in parameterized constructor, there is a hidden call to default constructor.
  super(s,f,a); //you can use super keyword only in the firstline. Because you'
e overriding the default call.

  System.out.println("Student class constructor 2");

   // SIN = s;
   // fullName = f;
   // address = a;

  studentID = sid;
  CGPA = c;

}
```

```java
public Person(long s, String f, String a){
  System.out.println("Person constructor 2");
  SIN=s;
  fullName=f;
  address=a;
}
```

Calling/Re-using parent class constructor can save us a lot of work in performing variable initializations, as demonstrated in today's class code.

Overriding

Can there be a method in both parent and child class at the same time ? Yes, but the child class method will ***override*** the parent class's method.

You must use the @Override annotation to let the compiler know that the overriding has been done on purpose and it's not a mistake. @Override is optional.

Person.java

```java
public void print(){
   //both private and public - methods and variables come from parent to children.
   System.out.println("SIN is "+SIN);
   System.out.println("FullName is "+fullName);
  System.out.println("Address is "+address);
 }
```

Student.java

```java
@Override
public void print(){
  //both private and public - methods and variables come from parent to children.
  System.out.println("Student Print");
  System.out.println("SIN is "+SIN);
  System.out.println("FullName is "+fullName);
   System.out.println("SID is "+studentID);
  setSIN(123455);
 }
```

Calling parent class methods

In the above code-snippet, we just saw that the print method in Student class is re-doing whatever the print method in Person is doing before getting to it's specialized work.

So, we are not using code-reuse properly. Instead we are replicating and doubling the amount of work. The smart way here would be to call the parent class's print method to print SIN, fullName and address and then sub-class's print can only print studentID and CGPA.

```java
@Override
public void print() {
  // both private and public - methods and variables come from parent to children.
  System.out.println("Student Print");
  super.print(); // Person.print() -> super keyword calls the properties of parents.
  System.out.println("SID is " + studentID);
  // setSIN(123455);
```

Multiple and Multi-Level Inheritance

Multiple inheritance is the concept where class A is inherited by class B and then class B is inherited by class C.

A->B->C

That is, multiple inheritance is kind of a parent-child-grandchild relationship.

Multiple inheritance is the concept when a class is derived from two or more parents. That is,

A&B->C

A and B together gives C.

Multiple inheritance is not allowed in Java. That is, a class cannot have more than one parent in Java.

A form of multiple inheritance can be achieved using Interfaces, but more on that in the upcoming lectures.