Inheritance

Learning Objectives

1. Learn inheritance.
2. Learn how to use **inheritance** to create **subclasses** that extend **superclasses**.

Agenda

- Warm-up
- Learn concepts: inheritance, superclass, subclass
- Learn inheritance Java syntax:
  - Classes
  - Constructors

Warm-up

Consider this scenario:

You are working for a university that wants you to write a program that will store information about various stakeholders (i.e. students, faculty, staff, etc).

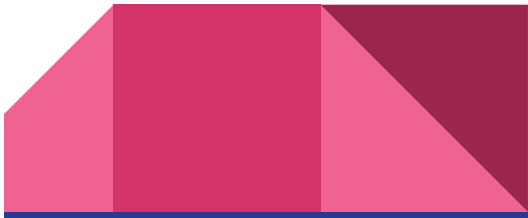
How would you go about solving this?

Designing classes: What do you think about this design?

```
class Student

String name
int id
double gpa
int year
```

```
class FacultyMember

String name
int id
int salary
String[]courses
String dept
```

```
class StaffMember

String name
int id
int salary
String role
```



Duplicated work

Once we write constructors, getters, and setters for all of those fields, we will have duplicated some of our code

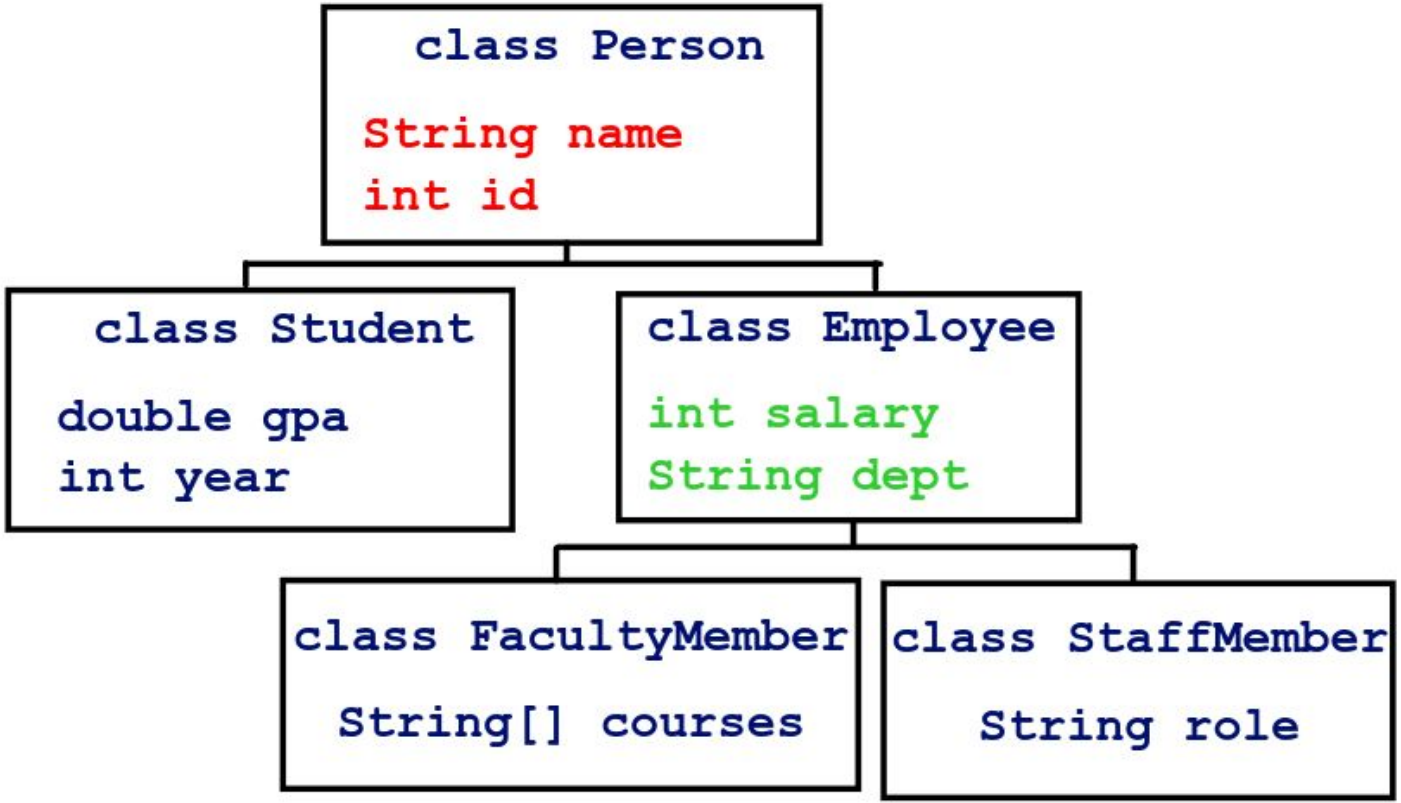
```
class Student
String name
int id
double gpa
int year
```

```
class StaffMember
String name
int id
int salary
String dept
String role
```

```
class FacultyMember
String name
int id
int salary
String dept
String[] courses
```

Solution

Generalize and re-use our code

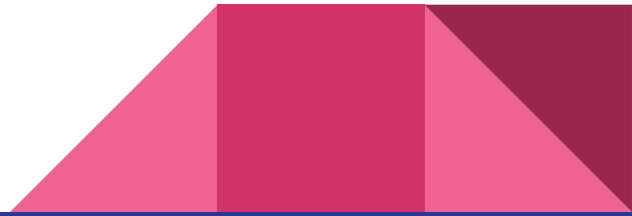


Inheritance

It is the process where **one class acquires the properties** (methods and fields) **of another**. Information is made manageable in a hierarchical order.

Inheritance allows to **create new classes** that are built **upon existing classes**.

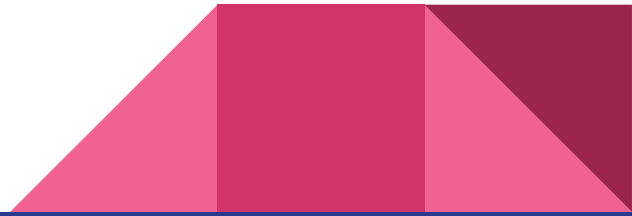
When you inherit from an existing class, you can **reuse methods and fields** of the parent class. Moreover, **new methods and fields can be added to a child class**.



Inheritance - Superclass, Subclass

A class that is derived from another class is called a **subclass** (also a derived class, extended class, or child class).

The class from which the subclass is derived is called a **superclass** (also a base class or a parent class).

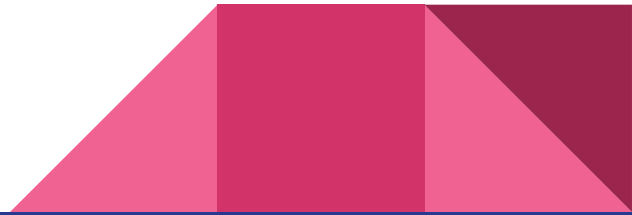


Inheritance - Object class

Object class is inherited by all the classes in Java (directly or indirectly). All its methods are available to all other classes in Java. Therefore, Object class behaves as a root of inheritance hierarchy in Java.

Excepting **Object**, which has no superclass, **every class has one and only one direct superclass** (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

Classes can be derived from classes that are derived from classes that are derived from classes, and so on.



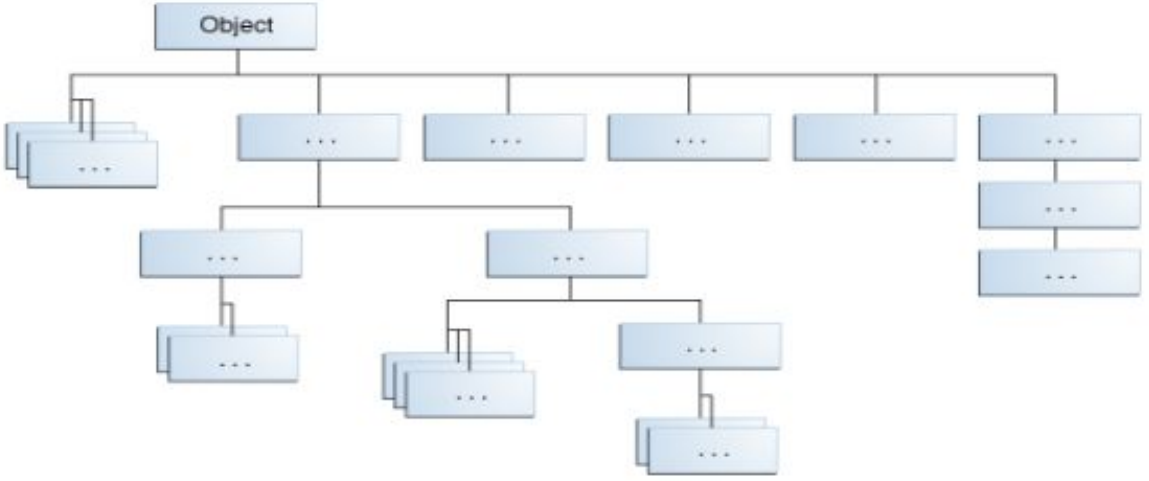
The Java Platform Class Hierarchy

The Object class, defined in the java.lang package, defines and implements behavior common to all classes.

Many classes derive directly from Object, other classes derive from some of those classes, and so on, forming a hierarchy of classes.

At the top of the hierarchy, **Object is the most general** of all classes.

Classes near the bottom of the hierarchy provide more **specialized behavior.**



All Classes in the Java Platform are Descendants of Object

Inheritance

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class.**You can reuse the fields and methods of the existing class** without having to write (and debug!) them yourself.

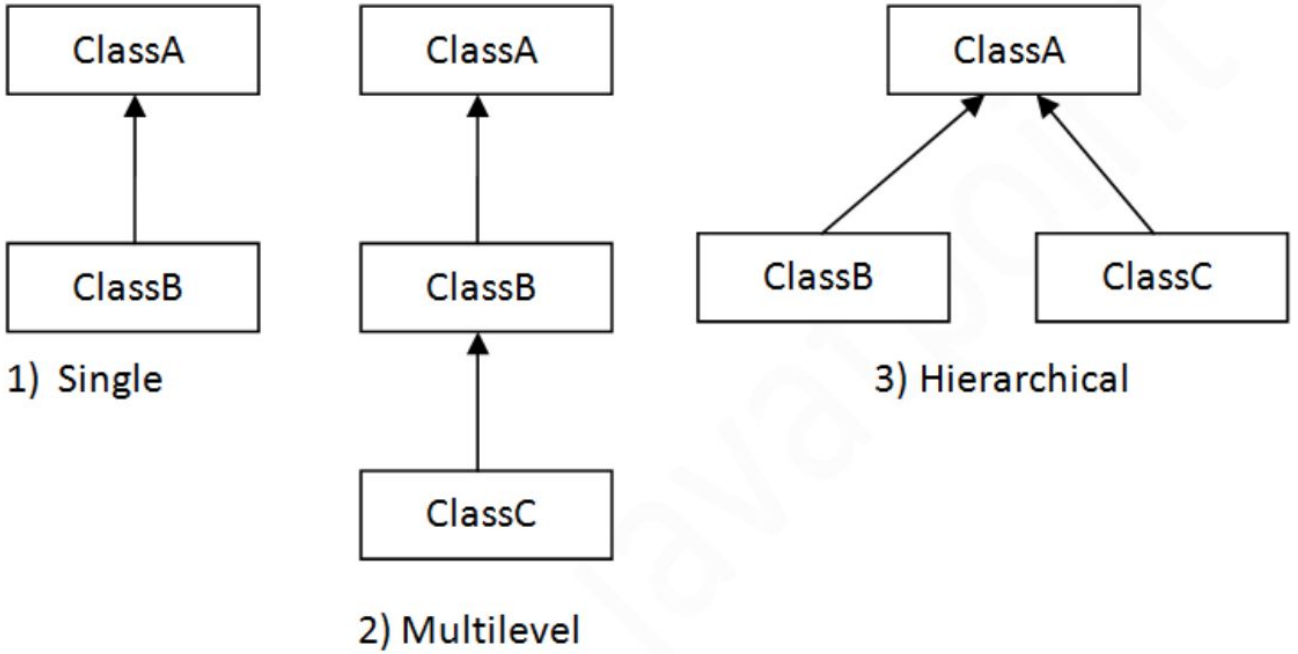
A subclass inherits all public the members (fields, methods) from its superclass. **Constructors are not members**, so they are not inherited by subclasses, but the **constructor of the superclass can be invoked** from the subclass.

Types of inheritance (supported by Java)

1) Single Inheritance: A derived class inherits from on one base class. Eg. class B inherits from class A.

2) Multilevel Inheritance: The class hierarchy is deeper that one level. Eg. class C inherits from class B that itself is inherited from class A.

3) Hierarchical Inheritance: A base class is specialized in many derived classes, which i turn are specialized into more derived classes.

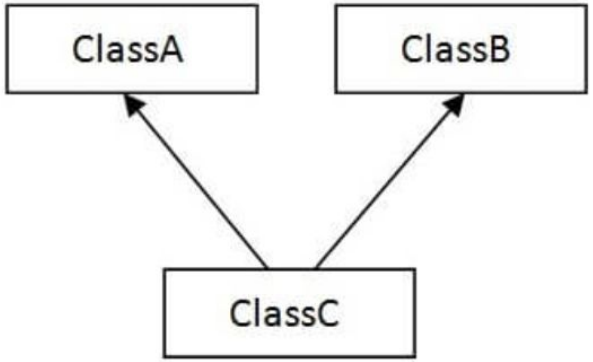


Types of inheritance (not supported by Java)

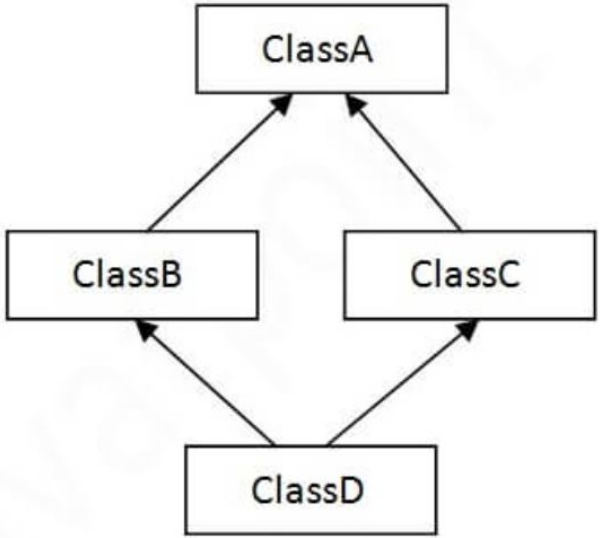
4. Multiple Inheritance: Class can inherit features from more than one parent object or parent class.

5. Hybrid Inheritance: A combination of simple, multiple inheritance and hierarchical inheritance.

To reduce complexity and simplify the language, multiple inheritance is not supported in Java.



4) Multiple



5) Hybrid

What You Can Do in a Subclass

A subclass inherits all of the ***public*** members of its parent. You can use the inherited members as is, replace them, hide them, or supplement them with new members:

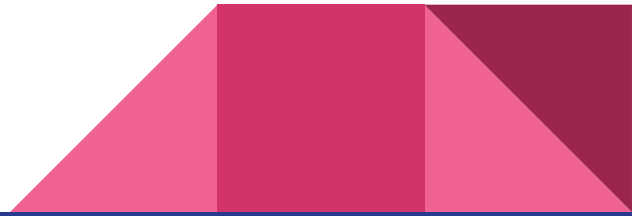
- The public inherited fields can be used directly, just like any other fields.
- You can declare a field in the subclass with the same name as the one in the superclass, thus *hiding* it (not recommended).
- You can declare new fields in the subclass that are not in the superclass.
- The inherited methods can be used directly as they are.
- You can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus *overriding* it.
- You can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.
- You can declare new methods in the subclass that are not in the superclass.
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

The **super** Keyword

The keyword `super` is very useful in allowing us to first execute the superclass method and then add on to it in the subclass.

Private Members in a Superclass

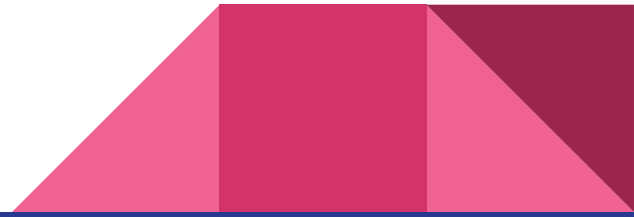
A subclass does not inherit the private members of its parent class. However, if the superclass has public methods for accessing its private fields, these can also be used by the subclass.



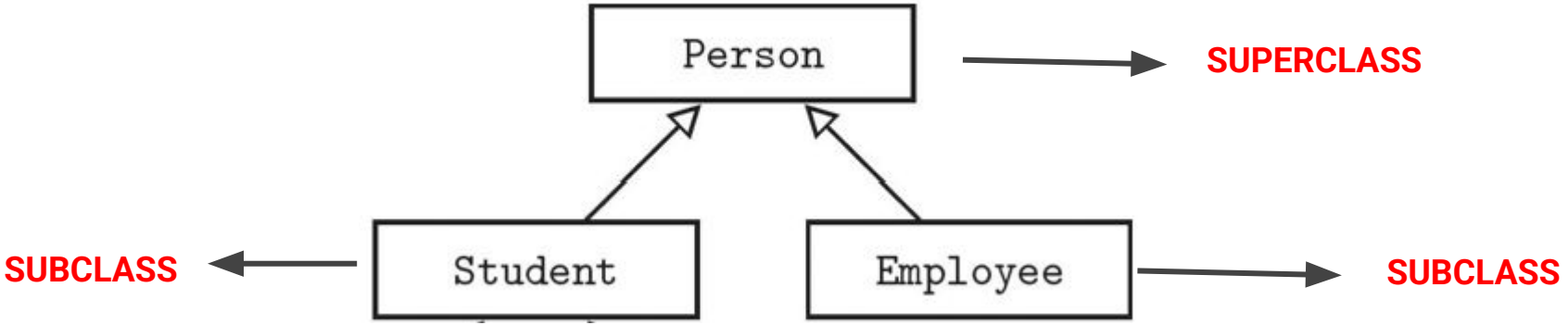
Java Syntax - Superclass and Subclass

```
public class Superclass
{
        //private instance variables
        //constructors
        //public methods
        //private methods
}
```

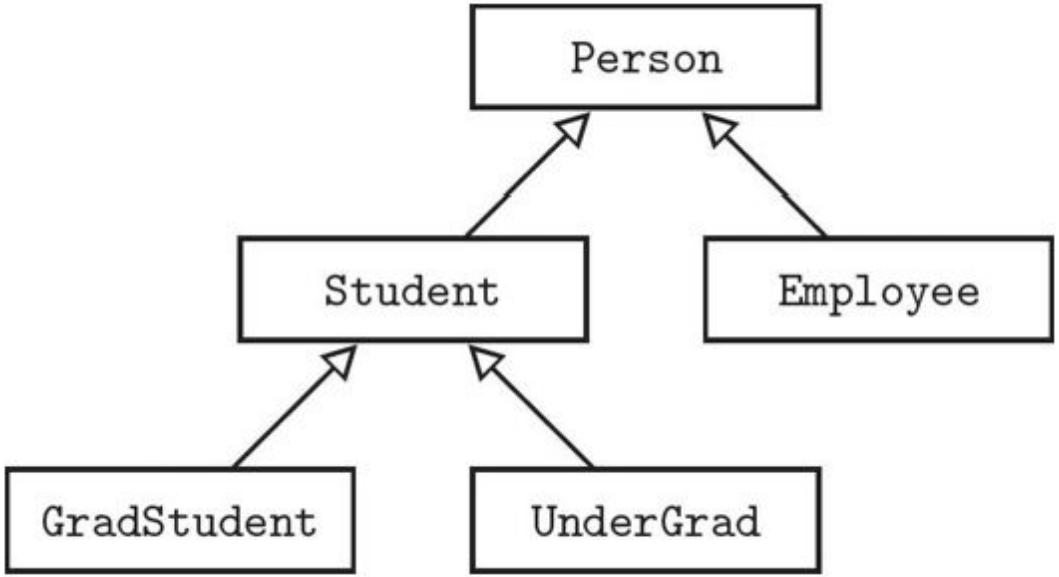
```
public class Subclass extends Superclass
{
        //additional private instance variables
        //constructors (Not inherited)
        //additional public methods
        //inherited public methods whose implementation is overridden
        //additional private methods
}
```



Inheritance - Example



Inheritance - Example

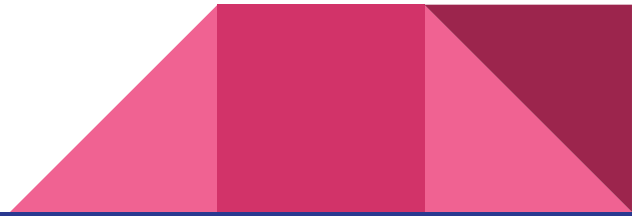


Demo Time!!!!

Let's write the following classes:

- Person
- Teacher
- Student

Which one is the super class? Which ones are the subclasses?



Did we meet the objectives?

1. Learn inheritance
2. Learn how to use **inheritance** to create **subclasses** that extend **superclasses**.