DePaul University

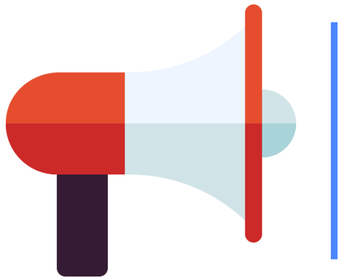
OOP Principles:

Inheritance

Object-oriented Software Development
SE 350– Spring 2021

Vahid Alizadeh

Week 3.1
April 13, 2021

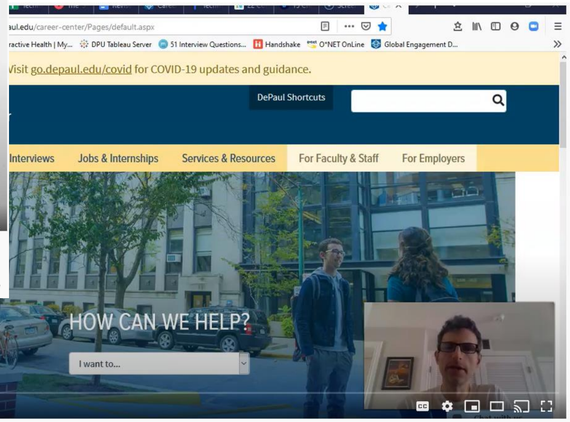


Announcements

DePaul Career Center Resources

- [Virtual advising services]
- [Online Career Library]
- [COVID-19 Career Resources]
- [DePaul Technology & Design Career Community LinkedIn group]

- **Some Video Resources:**
  - Tech & Design Career Community Intro:
  - 5 Tips for job searching during a pandemic
  - Library of 90 second videos



Tech & Design Career Community & Career Center Intro_final

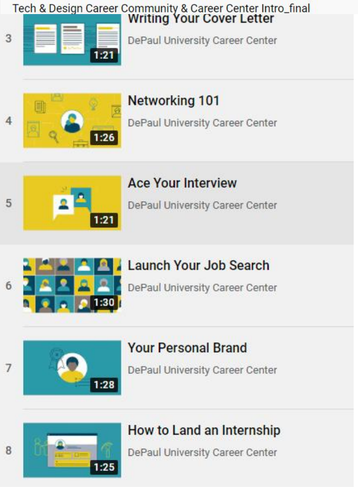
Career Library

14 videos • 1,110 views • Last updated on Jan 26, 2021

Welcome to the Career Video Library! We're creating these short videos to compliment the handouts and activity sheets in our Career Resource Library (bit.ly/CareerLibrary). Stay tuned, we'll be adding new videos weekly!

DePaul University Career Center

SUBSCRIBE



Assignment 1

Due: April 20, 2021

SE 350: OO Software Development

Assignment 1: OOP Basics

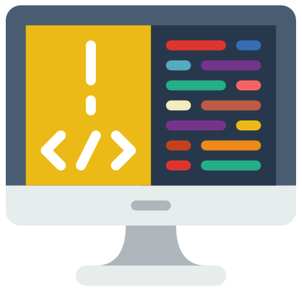
Instructor: Vahid Alizadeh

Email: v.alizadeh@depaul.edu

Quarter: Spring 2021

DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

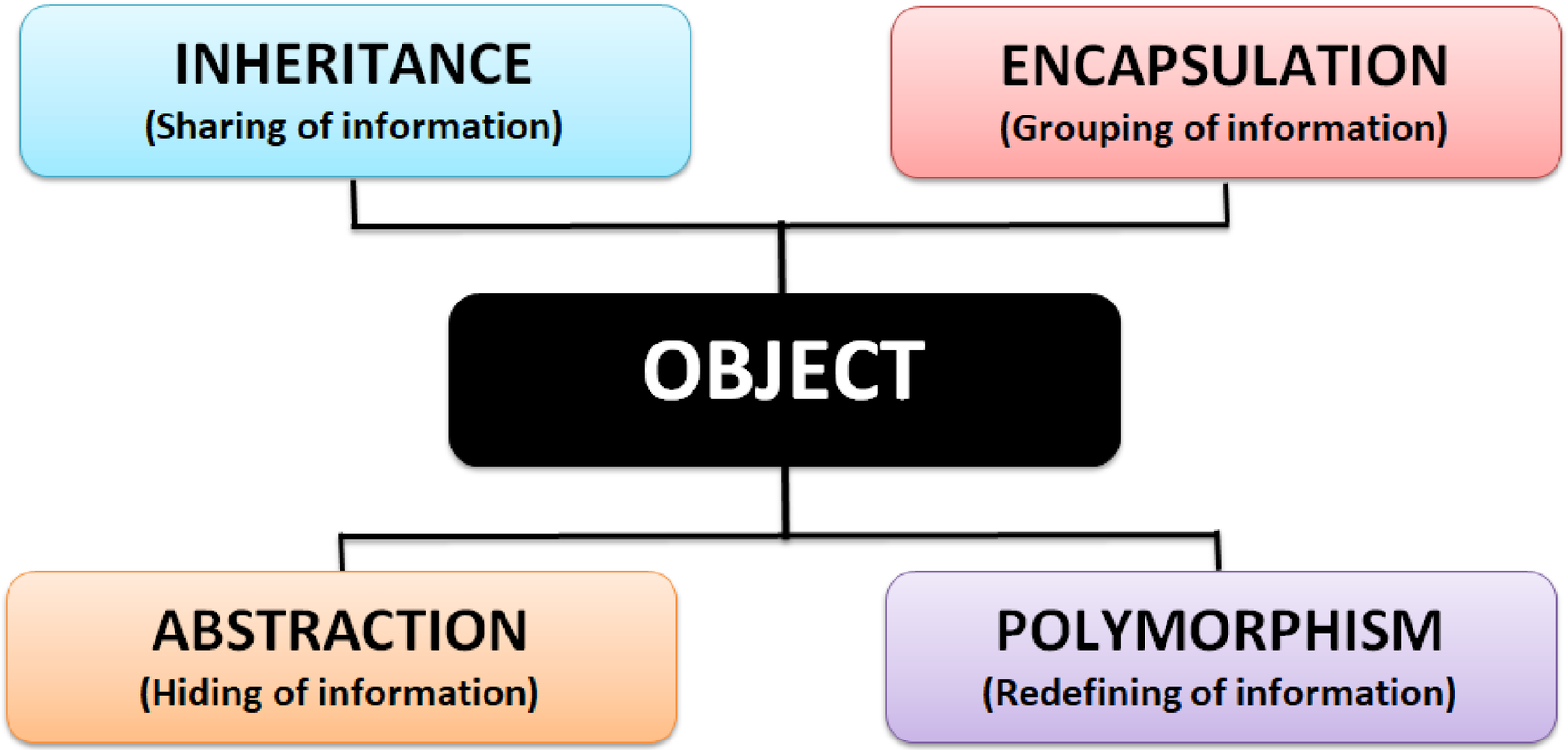
Last update: April 13, 2021

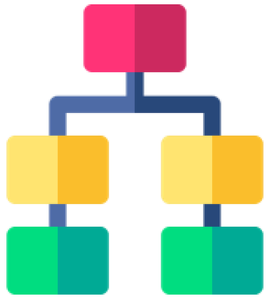


Object-oriented Programming

Principles

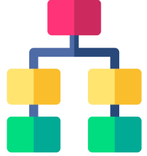
Principles of OOP





Object-oriented Programming Principles

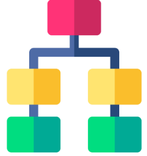
INHERITANCE



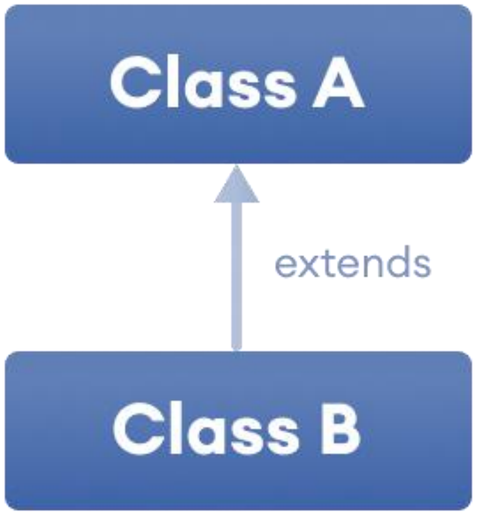
Inheritance

- Inheriting the common state and behavior
- Improve reusability & Reduce code redundancy
- Sub class vs. Super class
- *extends* keyword
- Inherits all the non-private members
- Example [package oopPrinciples.inheritance]
  - Employee and Manager

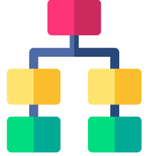
```java
public class Employee
{
    private Department department;
    private Address address;
    private Education education;
    //So on...
}

public class Manager extends Employee {
    private List<Employee> reportees;
}
```



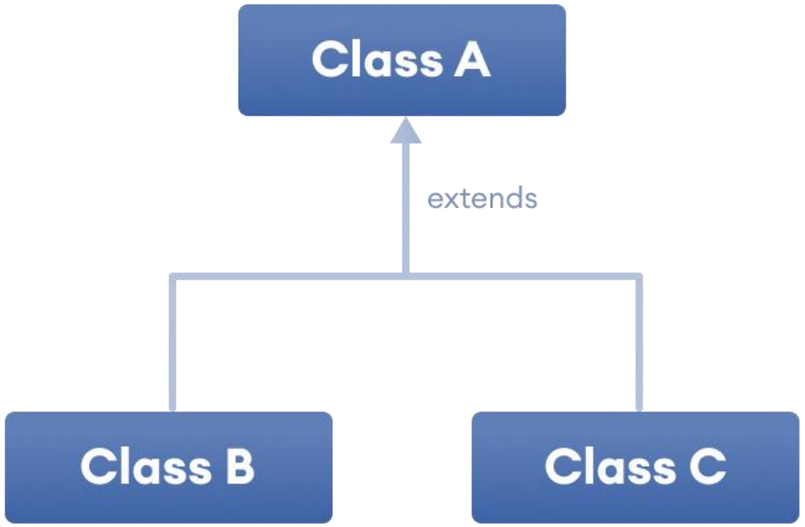
Types of Inheritance: Single Inheritance



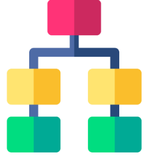

```
class Parent {
    //code
}

class Child extends Parent {
    //code
}
```



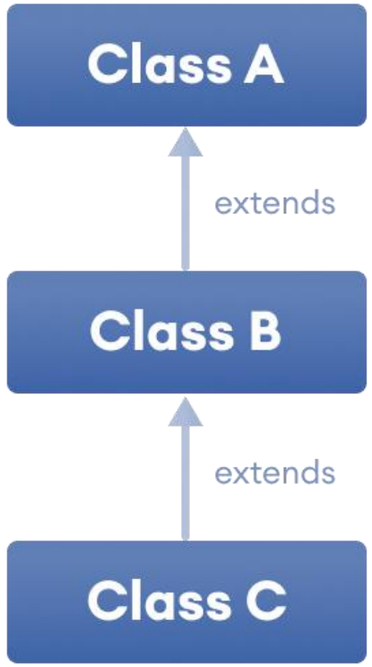
Types of Inheritance: Hierarchical Inheritance




```
class A {
  //code
}

class B extends A {
  //code
}

class C extends A {
  //code
}
```

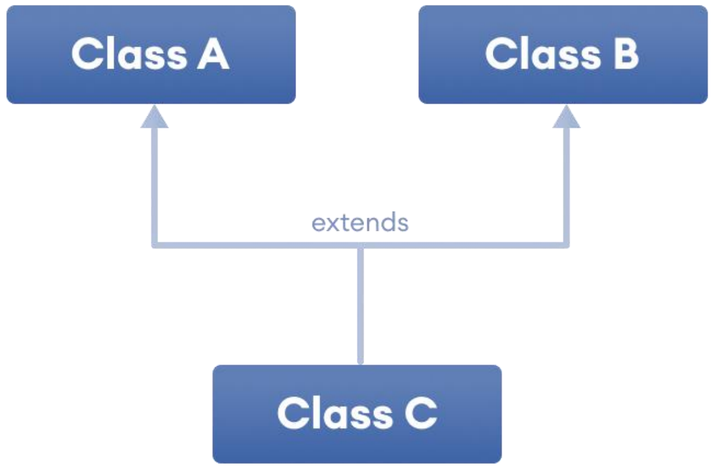


Types of Inheritance: Multi-level Inheritance

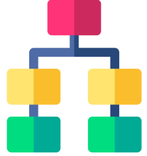



```
class A {

}

class B extends A {

}

class C extends B {

}
```

Types of Inheritance: Multiple Inheritance



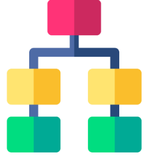

```
interface MyInterface1 {
  //code
}

interface MyInterface2 {
  //code
}

class MyClass implements MyInterface1, MyInterface2 {
  //code
}
```



Inheritance: Some Discussions

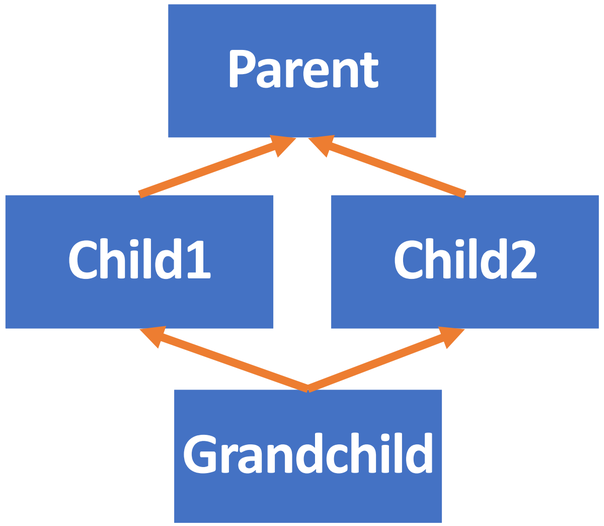
- **Object (in java.lang package) is the superclass for all classes.**
- **All members are inherited (except constructors)**
- **Child class can add new members but can't remove parent's members.**
- **Inheritance hierarchy is transitive**
- **Example** [package oopPrinciples.inheritance.discussion]
  - Demonstration of inheriting private members

```
//Demonstration of inheriting private members
class A {
    private int a;
}
class B extends A {
}
class Demo {
    public static void main(String[] args) {
        System.out.println("Demo Private members are also inherited***");
        B obB = new B();
        A obA = new A();
// This is a proof that a is also inherited. See the error message.
        System.out.println(obB.a);// Error:(16, 31) java: a has private access in
oopPrinciples.inheritance.discussion.A
        System.out.println(obB.b);// Error:(17, 31) java: cannot find symbol
//is not a field
        System.out.println(obA.a);// Error:(19, 31) java: a has private access in
oopPrinciples.inheritance.discussion.A
        System.out.println(obA.b);// Error:(20, 31) java: cannot find symbol
    }
}
```

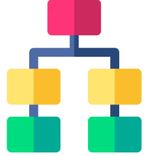


Inheritance: Some Discussions (1)

- **Why Java does not support multiple inheritance via class?**
  - Avoid ambiguity
  - Diamond problem
  - Example [package oopPrinciples.inheritance.discussion]



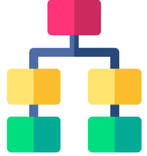

```java
class Parent {
    public void show() {
        System.out.println("I am in Parent");
    }
}
class Child1 extends Parent {
    public void show() {
        System.out.println("I am in Child1");
    }
}
class Child2 extends Parent {
    public void show() {
        System.out.println("I am in Child2");
    }
}
class GrandChild extends Child1,Child2// Error: Class can't extend multiple classes
{
    public void show() {
        System.out.println("I am in Grandchild");
    }
}
```



Inheritance: Some Discussions (2)

- **Is it true that all programming languages do not support multiple inheritance through classes?**
  - No, ex. C++ supports it.
- **Why does C++ support multiple inheritance through classes?**
- **Do we have Hybrid inheritance in Java?**
  - Yes! It is only a combination of two or more types of inheritance.
- **Which order constructors of the classes will be called?**
  - Order of calls follows the path from the parent to the child
  - Example [package oopPrinciples.inheritance.discussion]
- **How can you decide which class should be parent and child?**
  - "IS-A" relationship

```java
class ParentClass {
    ParentClass() {
        System.out.println("Inside Parent Constructor.");
    }
}
class ChildClass extends ParentClass {
    ChildClass() {
        System.out.println("Inside Child Constructor.");
    }
}
class GrandchildClass extends ChildClass {
    GrandchildClass() {
        System.out.println("Inside GrandChild Constructor.");
    }
}

class DemoConsOrder {
    public static void main(String[] args) {
        System.out.println("***Demo constructor calling order***");
        GrandchildClass grandChild = new GrandchildClass();
    }
}
```

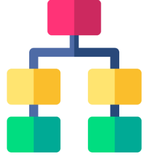


Accessing inherited parent class members

1. Parent class constructors

- *Super* keyword
- Must be:
  - made from child class constructor
  - first statement inside constructor

```
public class Manager extends Employee
{
    public Manager()
    {
        //This must be first statement inside constructor
        super();

        //Other code after super class
    }
}
```



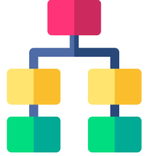
Accessing inherited parent class members

2. Parent class fields

- Access non-private fields using ***dot*** operator.
- Java fields cannot be overridden.
- Same name field case:
  - Decision based on the class of **Reference Type**

```
ReferenceClass variable = new ActualClass();
```

```
//Parent class
public class Employee
{
    public Long id = 10L;
}

//Child class
public class Manager extends Employee
{
    public Long id = 20L;   //same name field
}

public class Main {
    public static void main(String[] args)
    {
        Employee manager = new Manager();
        System.out.println(manager.id);     //Reference of type Employee

        Manager mgr = new Manager();
        System.out.println(mgr.id);     //Reference of type Manager
    }
}
//=============================================================================
//Output:

//10
//20
```



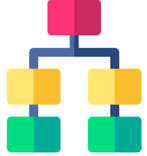
Accessing inherited parent class members

3. Parent class methods

- Method access uses the type of **actual object**

```
ReferenceClass variable = new ActualClass();
```

```
public class Employee
{
    private Long id = 10L;

    public Long getId() {
        return id;
    }
}

public class Manager extends Employee
{
    private Long id = 20L;

    public Long getId() {
        return id;
    }
}

public class Main
{
    public static void main(String[] args)
    {
        Employee employee = new Employee();     //Actual object is Employee Type
        System.out.println(employee.getId());

        Employee manager = new Manager();       //Actual object is Manager Type
        System.out.println(manager.getId());

        Manager mgr = new Manager();       //Actual object is Manager Type
        System.out.println(mgr.getId());
    }
}
//=============================================
//Output:

//10
//20
//20
```



Inheritance “Super” keyword Example

- **Example:** [package oopPrinciples.inheritance.discussion]
- **Why do you need to use *super* keyword?**
  - Avoid writing repeated code
  - Unable to access private members
- **Discussion**
  - Generalize usage of *super* keyword
  - “**super.member**”: member= instance field or method
  - Not invoking super class constructor >> call to the no-argument constructor of the super class
    - https://docs.oracle.com/javase/tutorial/java/IandI/super.html
  - Top of the class hierarchy >> java.lang.Object class
    - Clone(), toString(), notify(), hashCode(),…
  - *Super* keyword refers to the object of the immediate parent class.
  - *Super* keyword ~= *Base* keyword in C++ or C#

```
lass ParentCls {
    private int a;
    private int b;
    ParentCls(int a, int b) {
        System.out.println("I am in parent constructor.");
        System.out.println("Before setting,a="+ this.a);
        System.out.println("Before setting, b="+ this.b);
        System.out.println("Setting the values for instance variables a and b.");
        this.a = a;
        this.b = b;
        System.out.println("Now a="+ this.a);
        System.out.println("Now b="+ this.b);
    }
    void parentClsMethod() {
        System.out.println("I am a parent method.");
    }
}
class ChildCls extends ParentCls {
    private int c;
    ChildCls(int a, int b, int c) {
//      System.out.println("Before setting,c="+ this.c);
//      Error:Constructor call must be the first statement in a constructor
        super(a, b);
        System.out.println("I am in child constructor.");
        System.out.println("Before setting,c="+ this.c);
        this.c = c;
        System.out.println("Now c="+ this.c);
    }
    void childClsMethod() {
        System.out.println("I am a child method.");
        System.out.println("I am calling the parent method.");
        super.parentClsMethod();
    }
}

class DemoInheritanceSuperExample {
    public static void main(String[] args) {
        System.out.println("***DEMO: The uses of the 'super' keyword***");
        ChildCls sampleObj = new ChildCls(1, 2, 3);
        sampleObj.childClsMethod();
    }
}
```

Any Question

??????????????

Please Send Your Question or Feedback...

Top **New**