DePaul University

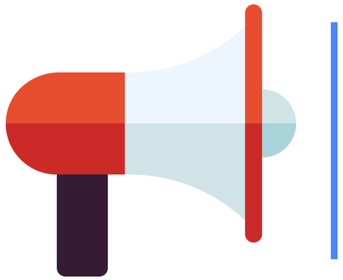
# OOP Principles:

## Abstraction | Encapsulation

**Object-oriented Software Development**
**SE 350– Spring 2021**

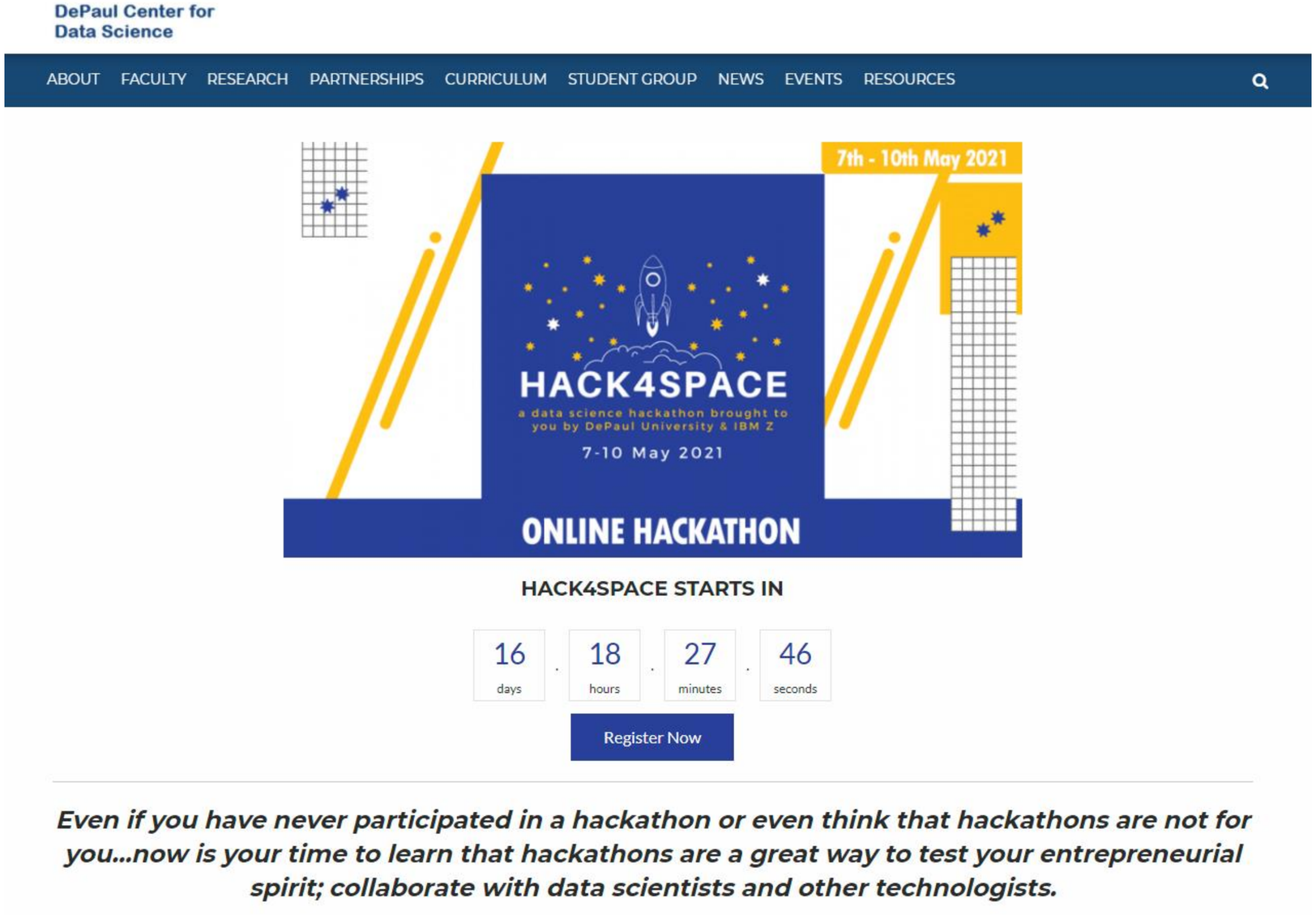
Vahid Alizadeh


Week 4.1
April 20, 2021



# Announcements

# HACK4SPACE


DePaul Center for Data Science
ABOUT FACULTY RESEARCH PARTNERSHIPS CURRICULUM STUDENT GROUP NEWS EVENTS RESOURCES
7th - 10th May 2021
HACK4SPACE
a data science hackathon brought to you by DePaul University & IBM Z
7-10 May 2021
ONLINE HACKATHON
HACK4SPACE STARTS IN
16 days
18 hours
27 minutes
46 seconds
Register Now
Even if you have never participated in a hackathon or even think that hackathons are not for you...now is your time to learn that hackathons are a great way to test your entrepreneurial spirit; collaborate with data scientists and other technologists.

# Assignment 2

## Due: February 27, 2021


**SE 350: OO Software Development**

**Assignment 2: OOP Principles and UML Class Diagram**

**Instructor:** Vahid Alizadeh

**Email:** v.alizadeh@depaul.edu

**Quarter:** Spring 2021

DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

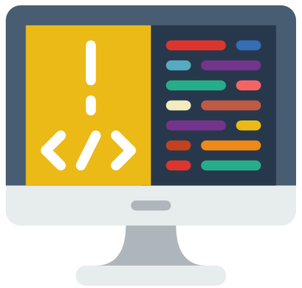
*Last update: April 20, 2021*

# Future Schedule

## Assignment 1 Solutions

## D2L / GitHub Repository

- ~~**Assignment 1:**~~
  - ~~Release: Week 3.1~~
  - ~~Due: Week 3.1~~
- **Assignment 2:**
  - Release: Week 4.1 (Today)
  - Due: Week 5.1
- **Mid Term Exam:**
  - Week 5.2



# Object-oriented Programming

## Principles

# Object-oriented Programming Principles

## ABSTRACTION

# Abstraction

- **Abstraction**
  - Real-life examples
    - Driving a car, Expanding or remodeling a house (Completing the incomplete work!)
  - Abstract classes = incomplete classes
  - Abstract methods = has declaration, no implementation
  - Subclass must finish the incomplete tasks
  - Used to define a generalized form shared between subclasses.
  - A class that contains at least one abstract method must be marked as an abstract class.
- **Example** [package oopPrinciples.abstraction]
  - An abstract class can also contain completed methods.
  - Subclass may or may not override these completed methods.

```
//A simple abstraction example
public class AbstractionExample {
}
abstract class AbstractClass {
    protected int myInt = 450;
    public abstract void showMe();
    public void completeMethod1() {
        System.out.println("I am from completeMethod1 in MyAbstractClass and I am complete.");
    }
    public void completeMethod2() {
        System.out.println("I'm the initial version of completeMethod2() in MyAbstractClass.I am
complete.");
    }
}
class CompleteClass extends AbstractClass {
    @Override
    public void showMe() {
        System.out.println("I'm 'completed/implemented version of showMe() in CompleteClass.");
        System.out.println("The value of myInt is:" + myInt);
    }
    @Override
//  It wants to override completeMethod1() in MyAbstractClass
    public void completeMethod1() {
        System.out.println("I'm the 'overrided' version of completeMethod1() in CompleteClass.");
    }
}
class DemoSimpleAbstractClass {
    public static void main(String Args[]) {
        System.out.println("***Demo Abstract classes.***\n");
        CompleteClass completeObj = new CompleteClass();
        completeObj.showMe();
        completeObj.completeMethod1(); // This will show that completeMethod1 is redefined in
CompleteClass.
        completeObj.completeMethod2(); // This will show the details of completeMethod2 defined in
AbstractClass.

        System.out.println("\n**Invoking methods through parent class reference now.**");
        AbstractClass abstractRef = new CompleteClass();
        abstractRef.showMe();
        abstractRef.completeMethod1();
        abstractRef.completeMethod2();
    }
}
```

# Abstraction Discussions

- You can implement the concept of runtime polymorphism here as well.
- Abstract class can contain fields.
- You can use any type of access modifier in an abstract class.
- Must mark a class as abstract even with only one abstract method.
- You cannot create objects from an abstract class.
- If a class extends an abstract class, it must implement all the abstract methods.
  - (Top Code Example)
- A concrete class is a class that is not abstract
- You cannot mark a method with both *abstract* and *final* keywords. Why?
- Constructors cannot be *final* or *abstract* or *static*. Why?
- You cannot reduce the visibility of an inherited method
  - (Bottom Code Example)
  - The access modifier of an overriding method must provide at least as much access as the overridden method itself.

```
abstract class AbstractClass {
  public abstract void inCompleteMethod1();
  public abstract void inCompleteMethod2();
}

abstract class child1 extends AbstractClass {
  //child class is implementing only one of the abstract methods.
  //So, the class is abstract again.
  @Override
  public void inCompleteMethod1()
  {
    System.out.println("Implementing the inCompleteMethod1()");
  }
}
```

```
abstract class IncompleteClass {
    public abstract void showMe();
}

class CompleteClass extends IncompleteClass {
    private void showMe() {
        System.out.println("I am complete.");
    }
}
```

# Interface

- **Interface**
  - A special type in *Java*
  - Contains method signatures to define specifications
  - Subtypes must follow the specifications
  - Declares ***What*** to implement, not ***How*** to implement
  - All methods are defined without body
  - May contain only final fields
  - Syntax: `interface MyInterface{..}`
  - Using interface, we can support multiple inheritance in Java.
- **Example** [package oopPrinciples.abstraction]

```java
//Demo of a simple interface
public class InterfaceExample {
}
interface MyInterface {
    void implementMe();
}
class MyClass implements MyInterface {
    public void implementMe() {
        System.out.println("MyClass is implementing the interface method implementMe().");
    }
}
class DemoInterface {
    public static void main(String[] args) {
        System.out.println("***Demo Simple Interfaces.***\n");
        MyClass myClassOb = new MyClass();
        myClassOb.implementMe();
    }
}
```

# Interface Discussions

- **Abstract class using interface**
  - The class that is using the interface must implement all the methods. If not, it is an abstract.
  - **Example** [package oopPrinciples.abstraction.discussion]
- **Extend and implement at the same time**
  - positional notation:
    - Extend before Implement

Extend & Implement Syntax

```
class ChildClass extends ParentClass implements Interface1,Interface2{...}
```

    - Following this design, the compiler knows about the parent class first and can point out any compilation errors in the parent class.

An abstract class inheriting interface

```
package oopPrinciples.abstraction.discussion;

public class InterfaceAbstractExample {
}
interface MyInterface{
    void show1();
    void show2();
}
//MyClass becomes abstract. It has not implemented show2() of MyInterface
//class MyClass implements MyInterface //error
abstract class MyClass implements MyInterface
{
    @Override
    public void show1() {
        System.out.println("MyClass is implementing the interface method show1().");
    }
// public abstract void show2();
}
class MySubClass extends MyClass
{
    @Override
    public void show2() {
        System.out.println("MySubClass is implementing the interface method show2().");
    }
}
class DemoInterfaceAbstract {
    public static void main(String[] args) {
        System.out.println("***Demo Interface and Abstract.***\n");

        MyInterface myOb = new MySubClass();
        myOb.show1();
        myOb.show2();
    }
}
```

# Interface Discussions

- **Multiple Inheritance using interface**
  - Example [package oopPrinciples.abstraction.discussion]
  - Interfaces' names are separated by commas
  - The method names can be the same and implementation class provides a common implementation.

Implementation of Multiple Interfaces

```
1  package oopPrinciples.abstraction.discussion;
2
3  public class MultipleInheritance {
4  }
5  interface MyInterfaceA {
6      void showA();
7  }
8  interface MyInterface5B {
9      void showB();
10 }
11
12 class MyClassMulti implements MyInterfaceA, MyInterface5B {
13     @Override
14     public void showA() {
15         System.out.println("Inside MyClass5,show5A() is completed.");
16     }
17     @Override
18     public void showB() {
19         System.out.println("Inside MyClass5,show5B() is completed.");
20     }
21 }
22
23 class DemoMultipleInheritance {
24     public static void main(String[] args) {
25         System.out.println("***Demo: Implementation of multiple interfaces.***\n");
26         MyClassMulti myClassMultiOb = new MyClassMulti();
27         myClassMultiOb.showA();
28         myClassMultiOb.showB();
29     }
30 }
```

# Interface Discussions

- **Interface can inherit but not implement another interface.**
  - **Example** [package oopPrinciples.abstraction.discussion]

- **Note:**
  - ✔ An interface **can** extend multiple interfaces.
  - ✘ A class **cannot** extend from multiple parent classes,
  - ✔ A class **can** implement multiple interfaces.

Interface can inherit another interface

```java
package oopPrinciples.abstraction.discussion;

public class InterfaceInheritClassExample {
}
interface InterfaceAA {
    void showInterfaceAAMethod();
}
interface InterfaceBB {
    void showInterfaceBBMethod();
}
//Interface extending another interface
interface InterfaceCC extends InterfaceAA, InterfaceBB {
    void showInterfaceCCMethod();
}
class MySampleClass implements InterfaceCC {
// Now MySampleClass needs to implement methods from InterfaceAA,InterfaceBB, and InterfaceCC

    @Override
    public void showInterfaceAAMethod() {
        System.out.println("MySampleClass has implemented the showInterfaceAAMethod() method.");
    }
    @Override
    public void showInterfaceBBMethod() {
        System.out.println("MySampleClass has implemented the showInterfaceBBMethod() method.");
    }
    @Override
    public void showInterfaceCCMethod() {
        System.out.println("MySampleClass has implemented the showInterfaceCCMethod() method.");
    }
}
```

# Default Methods in Interfaces

- **After Java 8, you can have a method with body in an interface.**
  - *Default* keyword
- **Example** [package oopPrinciples.abstraction.discussion]
- **You can override the default method in your implementation class. (Uncomment lines 24-28)**

Interface Default Method

```
1  package oopPrinciples.abstraction.discussion;
2
3  public class InterfaceDefaultMethod {
4  }
5  interface InterfaceWithDefault {
6      // Traditional interface method without a body.
7      void traditionalInterfaceMethod();
8
9      // A default method in the interface.
10     default void defaultMethod() {
11         System.out.println("Default implementation in the interface.");
12     }
13 }
14 class MyClassWithDefault implements InterfaceWithDefault {
15     @Override
16     public void traditionalInterfaceMethod() {
17         System.out.println("MyClassWithDefault is implementing the
   interface method");
18     }
19
20     public void sampleMethod(){
21         System.out.println("test");
22     }
23
24 //     Overriding the default method in an interface
25 //     @Override
26 //     public void defaultMethod() {
27 //     System.out.println("MyClassWithDefault is overriding the default
   interface method.");
28 //     }
29 }
30 class DemoInterfaceDefaultMethod {
31     public static void main(String[] args) {
32         System.out.println("***Demo using default methods in an
   interface***\n");
33         InterfaceWithDefault interfaceOb = new MyClassWithDefault();
34         interfaceOb.traditionalInterfaceMethod();
35         interfaceOb.defaultMethod();
36     }
37 }
```