OntarioTech UNIVERSITY

Tutorial #5 Object Oriented Design

SOFE 2710U Object Oriented Programming and Design

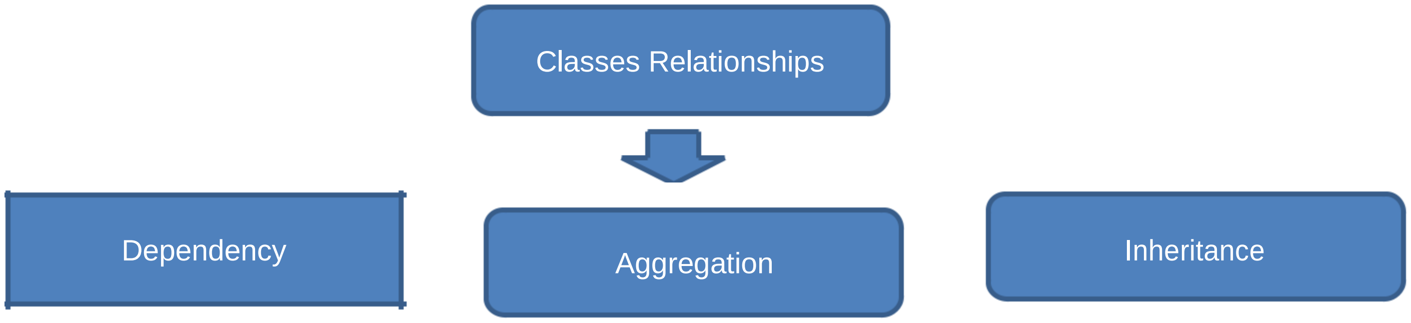
FALL 2019

Objective:

In this tutorial, you will learn Class relationships, this reference, enumerated types and method overloading

Class Relationship

In Java, classes may have various type of relationships to each other as shown:



I. Dependency

Dependency exists when one class is dependent on the other, usually by invoking the methods of the other. This may happen between objects of the same class. For example, the Concat method of the string class takes in as a parameter string object.

str3 =str1.concat(str2)

```
//----------------------------------------------------------------
  //  Multiplies this rational number by the one passed as a
  //  parameter.
  //----------------------------------------------------------------
  public RationalNumber multiply(RationalNumber op2)
  {
     int numer = numerator * op2.getNumerator();
     int denom = denominator * op2.getDenominator();
     return new RationalNumber(numer, denom);
  }
```

II. Aggregation

Aggregation is an object that is composed of other Objects. Aggregation represents **Has A**

relationship. For Example: Consider two classes Student and Address:

A student has an address.

An aggregate object comprises references of other objects as instance data.

(, a Student object is composed, in part, of Address objects)

```
public class Student

{

   ...

   private Address homeAddress, schoolAddress;

        ...

}
```

The this Reference

The this keyword is used to refer to the current object and is always a reference to the object on which method was invoked

```
public Box (double width, double height, double depth)
{
        this.width = width;
        this.height = height;
        this.depth = depth;
}
```

Interfaces

A Java interface is a collection of abstract methods and constants. An abstract method is a method header without a method body. An interface is used to establish a set of methods that a class will implement. If a class declares that it implements an interface, it must define all methods in the interface.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
```

Enumerated Types

An enumerated type definition can be more interesting than a simple list of values. Because they are like classes, we can add additional instance data and methods. We can define an enum constructor as well.

Method Overloading

If the class has one or method with the same name but different parameters, this refers to as method overloading. Method overloading always occur within the same class. The signature of each overloaded method must be unique. The signature includes the number, type, and order of the parameters. The compiler is able to determine which method is being invoked by analyzing the parameters as shown in the Example:

```
class Calculate
{
  void sum (int a, int b)
  {
    System.out.println("sum is"+(a+b));
  }
  void sum (float a, float b)
  {
    System.out.println("sum is"+(a+b));
  }
  Public static void main (String[] args)
  {
    Calculate cal = new Calculate();
    cal.sum (8,5);     //sum(int a, int b) is method is called.
    cal.sum (4.6f, 3.8f); //sum(float a, float b) is called.
  }
}
```

Output

```
Sum is 13
Sum is 8.4
```

Problems:

1. Write a method called average that accepts two integer parameters and returns their average as a floating-point value.
2. Overload the average method of question 1 such that if three integers

are provided as parameters, the method returns the average of all three.

3. Overload the average method of question 1 to accept four integer parameters

and return their average.

4. Files Question.java, Complexity.java, and MiniQuiz.java contain the classes. These classes demonstrate the use of the Complexity interface; class Question implements the interface, and class MiniQuiz creates two Question objects and uses them to give the user a short quiz.

Save these three files to your directory and study the code in MiniQuiz.java. Notice that after the Question objects are created, almost exactly the same code appears twice, once to ask and grade the first question, and again to ask and grade the second question. Another approach is to write a method askQuestion that takes a Question object and does all the work of asking the user the question, getting the user's response, and determining whether the response is correct. You could then simply call this method twice, once for q1 and once for q2. Modify the MiniQuiz class so that it has such an askQuestion method, and replace the code in main that asks and grades the questions with two calls to askQuestion.