CS 5004:
Object Oriented Design and Analysis
Spring 2024

Tamara Bonaci

t.bonaci@northeastern.edu

Agenda

- Inheritance and “is-a” relationship
- Composition and “has a” relationship
- Abstract classes
- Interfaces

REVIEW

Review: Inheritance and “Is a” Relationship

- Inheritance - set of classes connected by an ‘is-a’ relationships
- ‘Is-a’ relationship - hierarchical connection where one category can be treated as a specialized version of another
  - Example 1:
    - Every student is a person
    - Every ALIGN student is a student
  - Example 2:
    - Every pepper is a vegetable
    - Every bell pepper is a pepper
    - Every banana pepper is a pepper

Review: Class Inheritance

- Many programming languages (Java, C++, C#) provide a direct support for is-a relationship through class inheritance

- Class inheritance - new class extends existing class
  - Original/Extended class (also known as base class or super class)
  - New/Extending class (also known as derived class or subclass)

- Rules for derived classes (subclasses):
  - Derived class automatically inherits all NON-private instance variables and methods of the base class
  - Derived class can add additional methods and instance variables
  - Derived class can provide different versions of inherited methods

- Note: in Java a class can extend ONLY one class

Review: Class Inheritance

Derived class automatically inherits all NON-private instance variables and methods of the base class

```
public class Person {
  protected String firstName;
  protected String lastName;

  public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}
```

Parent class with protected fields

```
public class Student extends Person {
  private String studentID;

  public Student(String firstName, String lastName, String studentID) {
    super(firstName, lastName);
    this.studentID = studentID;
  }

  public void printStuff(){
    System.out.println(this.firstName);
    System.out.println(super.firstName);
  }
}
```

Child class –direct access to the parent's protected fields

Review: Class Inheritance

Derived class automatically inherits all NON-private instance variables and methods of the base class

```
public class Person {
  private String firstName;
  private String lastName;

  public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}

public class Client extends Person {
  private String clientID;

  public Client(String firstName, String lastName, String studentID) {
    super(firstName, lastName);
    this.clientID = studentID;
  }

  public void printStuff(){
    System.out.println(this.getFirstName());
    System.out.println(super.getLastName());
  }
```

Parent class with private fields

Child class – no direct access to the parent's private fields

Review: Composition and “Has a” Relationship

- Composition - set of classes connected by an ‘has-a’ relationships
- ‘Has-a’ relationship – a relationship where one class can use the functionality of another class by using an instance of that class
- Example 1:
  - Every person has a name
  - Every person has a date of birth
- Example 2:
  - Every vehicle has a make
  - Every vehicle has a model
  - Every vehicle has a manufacturing year

ABSTRACT CLASSES

Abstract Classes

- Abstract class:
  1. Cannot be instantiated
  2. Typically used as a base class for subclasses
  3. Can contain **abstract** method – methods that are not fully implemented
  4. Can be extended by subclasses to create a full implementation

Abstract Classes - Example

```
public abstract class AbstractShape implements Shape {
    protected Point2D pin;

    public AbstractShape (Point2D point) {
        this.pin = point;
    }
}
```

Abstract parent class

```
public class Circle extends AbstractShape {
    private Integer radius;

    public Circle (Point2D pin, Integer radius) {
        super(pin);
        this.radius = radius;
    }
}
```

Concrete child class

INTERFACES

Interface

- What is an interface?
  - Set of method declarations (signatures) representing a common public behavior of a class
  - Contract /protocol of what the classes can (should) do

- Think of an interface as a way to provide a view of your objects that shows only the relevant methods that can be used on your objects from this view

Interface

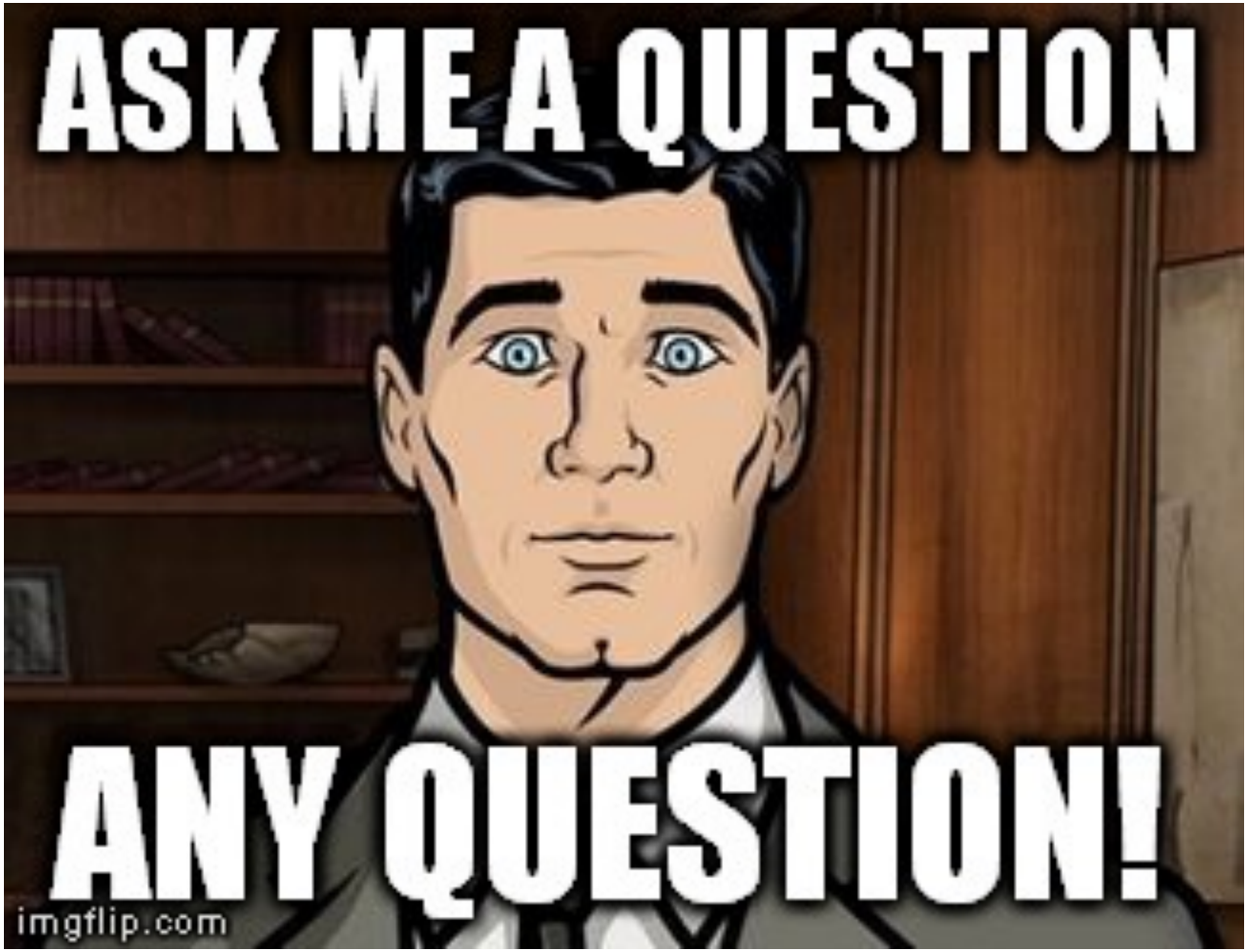
- What is an interface?
  - Set of method declarations (signatures) representing a common public behavior of a class
  - Contract /protocol of what the classes can (should) do
- Some rules for interfaces:
  - No constructor – you cannot directly create an instance of an interface
  - No fields in an interface
  - Everything in the interface is public by default
  - No method implementations – only method signatures
  - A class implements an interface by explicitly declaring in its header the name of the interface using keyword implements
  - Class that implements an interface must implement its complete behavior (all of the declared methods)

Abstract Classes and Interfaces

Abstract class	Interface
A class can extend **at most one** superclass (abstract or concrete)	A class can implement **any number** of interfaces
Includes instance variables	No instance variables (in Java 8)
Wider range of modifiers (`private, public, protected`)	All methods has `public` access modifier by default
Can specify constructors, which subclasses can invoke with keyword `super` (abstract classes still cannot be instantiated)	No constructors (interfaces cannot be instantiate)
Use: to abstract out common states and behavior among children classes	**Use:** a contract, specifying public behavior

[Table credit: Dr. Maria Zontak]

Your Questions



[Meme credit: imgflip.com]