CISC 3115 EWQ6

Inheritance

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recall we discussed
  - Relationships of classes
    - Association (Composition, and Aggregation)
    - There are more!

- Inheritance
  - Superclass/supertype, subclass/subtype

- Inheritance and constructors in Java

- Inheritance and instance methods in Java

- The Object class in Java

Class and Type

- A class defines a type, and often models a set of entities (or objects)

- Example: to build a system for managing business at Brooklyn College, we consider

  - People, a set of individuals (objects), modeled as a class that captures the essence of the set of objects

  - We have a _type_ of objects, called People

People at Brooklyn College

Subtypes

- Some people at Brooklyn are different from the others in some way

- Professors and students are also types of Brooklyn College People, but professors and students are also People – they are subtypes of People

| Professors | Students |

People at Brooklyn College

Type Hierarchy

- We have a hierarchy of types! They share a common set of characteristics and behavior, and also differ in some ways

    - What do Students and Professors have in common?

    - How are Students and Professors different?
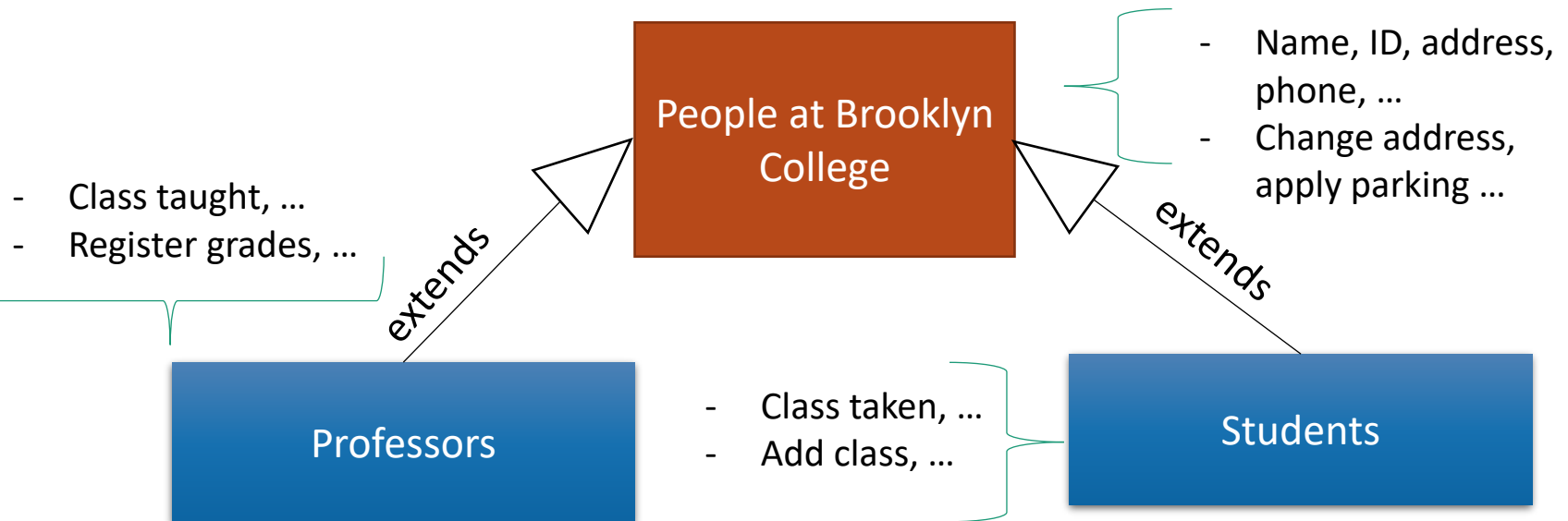
What's in Common?

- What characteristics (attributes) and behavior (actions) do People at Brooklyn College have in common?

  - Characteristics (attributes, fields, or states): name, ID, address, email, phone, …

  - Behavior (actions, functions, or methods): change address, apply parking, …

What's Special?

- What's distinct about students?

  - Characteristics (attributes, fields, or states): classes taken, tuition and fees, …

  - Behavior (actions, functions, or methods): add class, drop class, pay tuition, …

- What's distinct about professors?

  - Characteristics (attributes, fields, or states): course taught, rank, title, …

  - Behavior (actions, functions, or methods): register grade, apply promotion, …
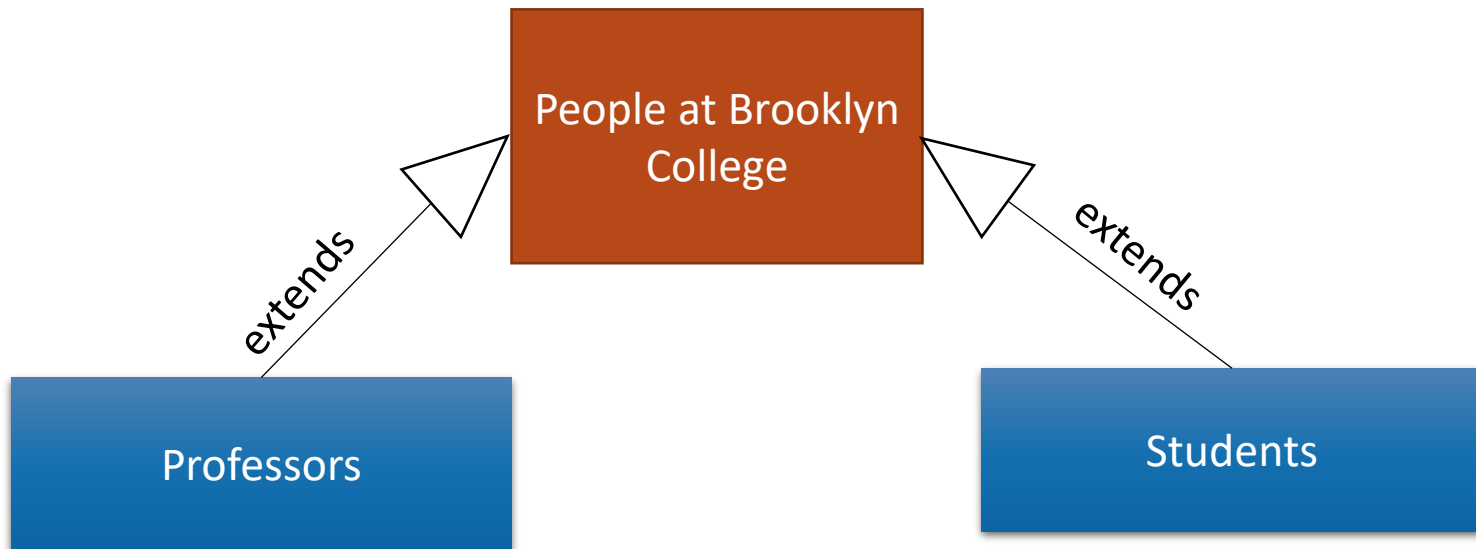
Inheritance & Type Hierarchy

- A subtype (child) inherits characteristics (data fields & methods) and behavior (actions) of its super/base type (parent)

- Class taught, …
- Register grades, …

People at Brooklyn College

- Name, ID, address, phone, …
- Change address, apply parking …

Professors

extends

extends

- Class taken, …
- Add class, …

Students

Remark: Graphing Type Hierarchy

- UML class diagrams

```
                    ┌──────────────────┐
                    │ People at Brooklyn│
                    │      College      │
                    └──────────────────┘
           extends                      extends
    ┌──────────────┐              ┌──────────────┐
    │  Professors  │              │   Students   │
    └──────────────┘              └──────────────┘
```
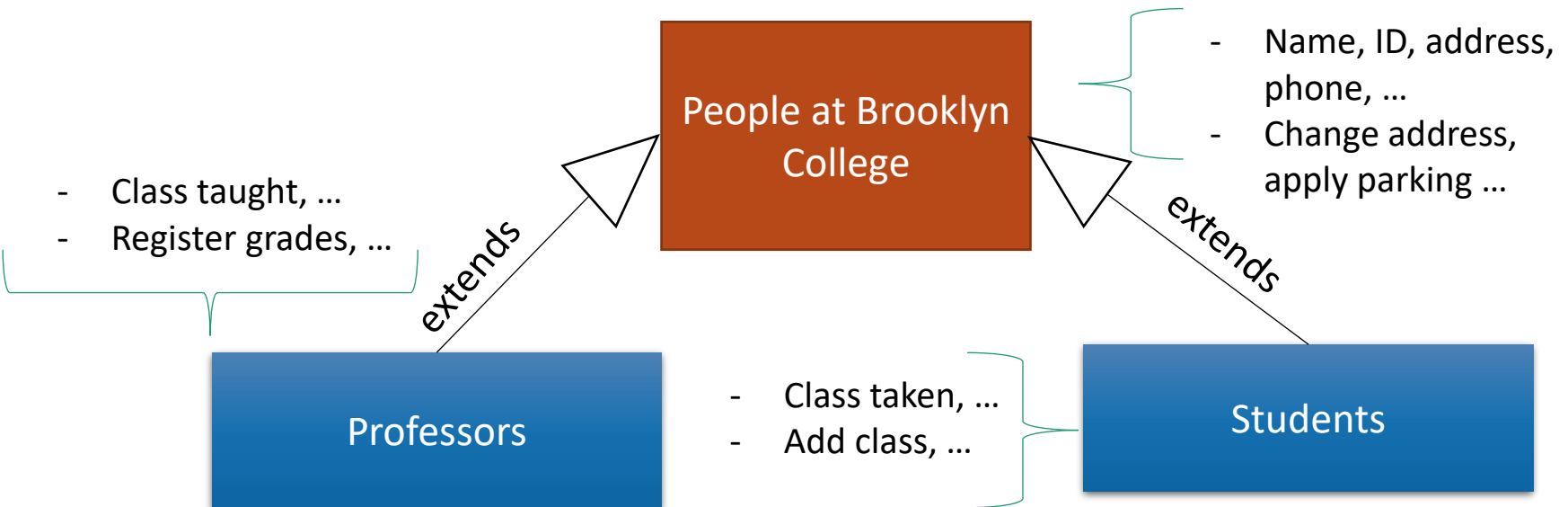
Terms of Choice

- Terms
  - Super type, Super class
  - Base type, Base class
  - Parent type, parent class
  - Child type, child class
  - Subtype, subclass
  - …
- In Java, we sometimes consider "type" and "class" are slightly different
  - In Java, a pure abstract class is called an "interface" (to be discussed in the future)

Example: Realizing the Type Hierarchy

- Classes: Person, Student, Professor

Super Type (Super Class): Person

public class Person {

        private String name;

        private String id;

        private String address;

        public Person(String name, String id, String address) {

                this.name = name; this.id = id; …

        }

        public void changeAddress(String address) { …   }

… }

Subtype (Subclass): Student

public class Student <u>extends</u> Person {

        public final static int MAX_NUM_COURSES = 10;

        private String[] classesTaken;

        public Student(String name, String id, String address) {

                <u>......</u> // initializing inherited data fields

                classesTaken = new String[MAX_NUM_COURSES];

        }

        public void haveTakenClass(String className) { … }

        public void showClassesTaken() { … }

…}

Subtype (Subclass): Professor
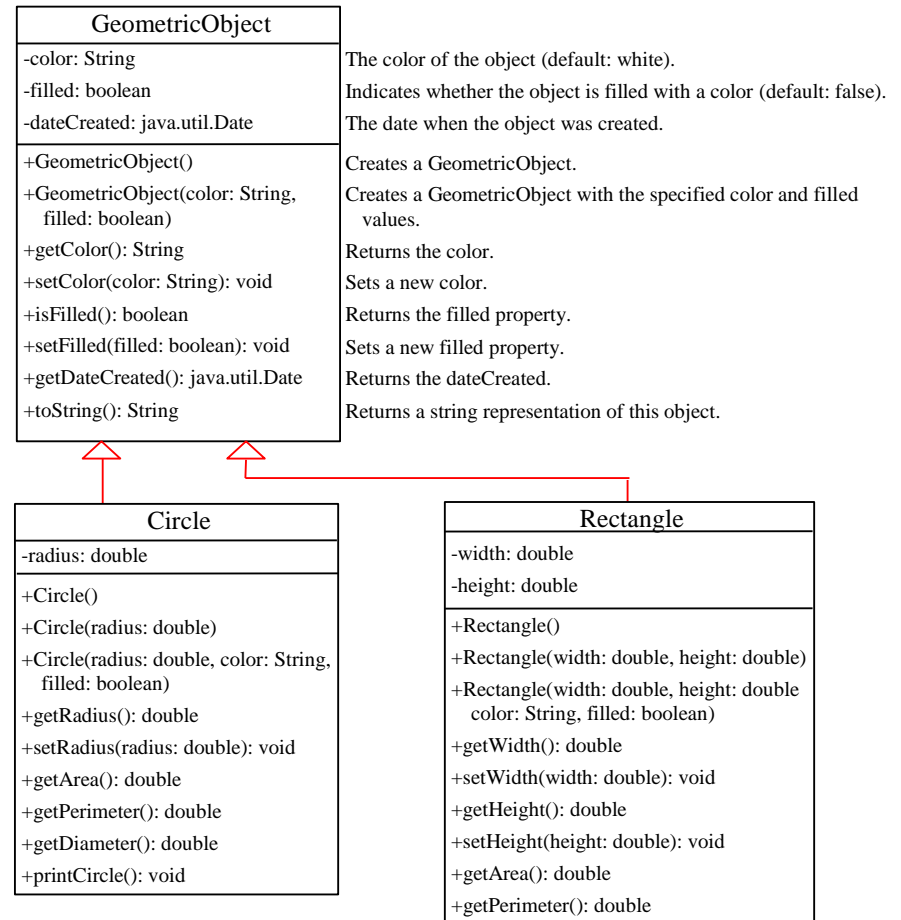
```
public class Professor extends Person {

        public final static int SABATTICAL_LEAVE_INTERVAL = 7;

        private int yearStarted;

        public Professor(String name, String id, String address, int yearStarted) {

                ......  // initializing inherited data fields

                this.yearStarted = yearStarted;

        }

        public void applySabbatical(int applicationYear) { …

        }

…}
```

Questions

- Concepts
  - Type, subtype, class, subclass
  - Inheritance

UML Diagram and Type Hierarchy

- UML diagram for showing class hierarchy

- Example: GeometricObject, Circle, Rectangle

GeometricObject	
-color: String	The color of the object (default: white).
-filled: boolean	Indicates whether the object is filled with a color (default: false).
-dateCreated: java.util.Date	The date when the object was created.
+GeometricObject()	Creates a GeometricObject.
+GeometricObject(color: String, filled: boolean)	Creates a GeometricObject with the specified color and filled values.
+getColor(): String	Returns the color.
+setColor(color: String): void	Sets a new color.
+isFilled(): boolean	Returns the filled property.
+setFilled(filled: boolean): void	Sets a new filled property.
+getDateCreated(): java.util.Date	Returns the dateCreated.
+toString(): String	Returns a string representation of this object.

Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void

| Rectangle |
|---|
| -width: double |
| -height: double |
| +Rectangle() |
| +Rectangle(width: double, height: double) |
| +Rectangle(width: double, height: double color: String, filled: boolean) |
| +getWidth(): double |
| +setWidth(width: double): void |
| +getHeight(): double |
| +setHeight(height: double): void |
| +getArea(): double |
| +getPerimeter(): double |

Exercise (Part 1 of 3)

- Complete the following,
  - Implement 3 classes: Shape, Circle, and Rectangle with *minimal coding (don't write more than asked)*
    - The Shape class is the superclass of the Circle and Rectangle class
    - Shape objects have a name. We add the name data field to the Shape class
    - We add a getName():String method to the Shape class
    - Write a ShapeClient class and create a Shape, a Circle, and a Rectangle object, and print out their names.
    - Make sure you can compile your classes
  - We shall do more with these classes (later)

Constructors

- Let us consider

  - Circle c = new Circle();

- Are superclass's constructor inherited?

  - No. They are not inherited.

  - They are invoked explicitly or implicitly.

  - Explicitly using the super keyword.

Constructors

- Let us consider

  Circle c = new Circle();

- Are superclass's Constructor Inherited?

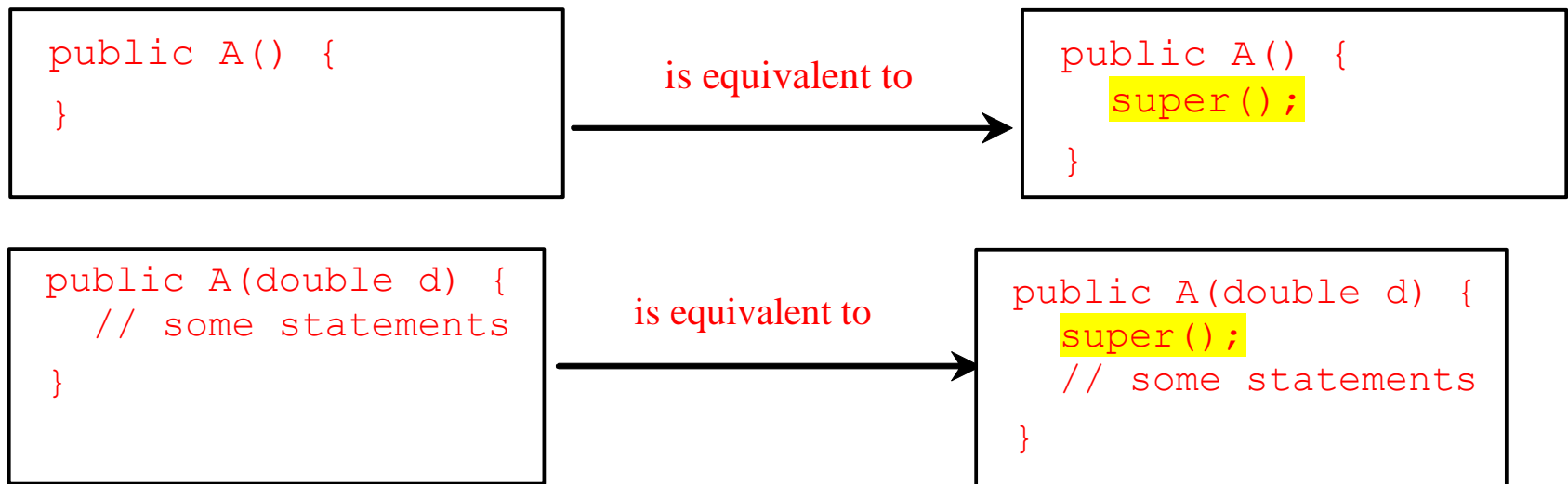- In other words, how are the data fields initialized?
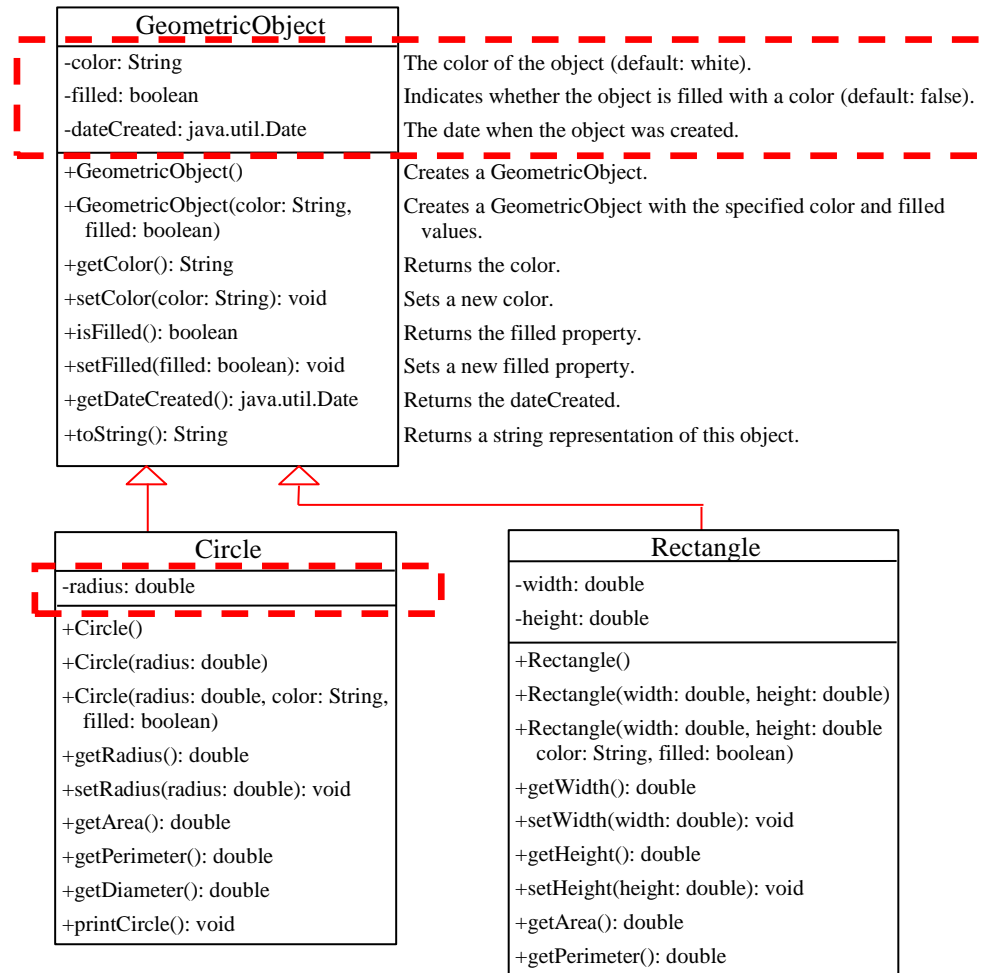
Constructors in Super- and Sub-Classes

- Are superclass's Constructor Inherited?

  - No. They are not inherited, but one is always invoked

  - They are invoked <u>explicitly</u> or <u>implicitly</u>.

  - <u>Explicitly</u> using the super keyword

  - <u>Implicitly</u> *the superclass's no-arg constructor is automatically invoked if the keyword <u>super</u> is not explicitly used.*

Implicit Invocation of Superclass's Constructor

- A superclass's constructor is always invoked even if it isn't invoked explicitly using super.

- Which constructor is invoked implicitly?

```
public A() {

}
```
is equivalent to
```
public A() {
    super();

}
```

```
public A(double d) {
    // some statements
}
```
is equivalent to
```
public A(double d) {
    super();
    // some statements

}
```

Explicit Invocation of Superclass's Methods

- super refers to the superclass

- Use it

  - To call a superclass constructor

    - Java requires that the statement that uses the keyword <u>super</u> appear first in the constructor.

  - To call a superclass method

Constructor Chaining

- Invocation of superclass's constructor (along the inheritance chain)

- Example

  - Consider classes: Person, Employee, Faculty

CUNY | Brooklyn College
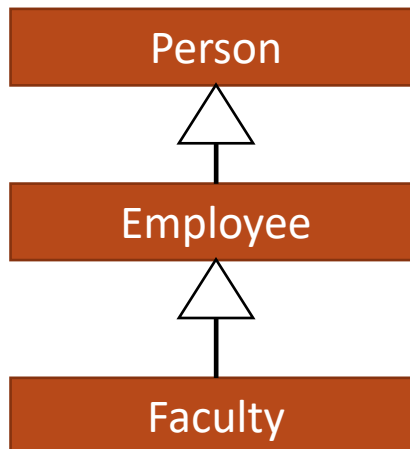
Constructor Chaining: Example

```java
class Person {

 public Person() {

   System.out.println("(1) Person's no-arg constructor is invoked");

 }

}


 class Employee extends Person {

 public Employee() {

   this("(2) Invoke Employee's overloaded constructor");

   System.out.println("(3) Employee's no-arg constructor is invoked");

 }
```

```java
class Employee(String s) {

   System.out.println(s);

 }

}


class Faculty extends Employee {

 public static void main(String[] args) {

   new Faculty();

 }

 public Faculty() {

   System.out.println("(4) Faculty's no-arg constructor is
invoked");

 }

}
```

Discussion: No-Arg Constructor

- Is there an error in the code below, and why?

```
public class Apple extends Fruit {

}


public class Fruit {
  public Fruit(String name) {
    System.out.println("Fruit's constructor is invoked");
  }
}
```

CUNY | Brooklyn College

Questions?

- Constructors in superclass

  - Explicit and implicit invocation

- Constructor chaining

Exercise (Part 2 of 3)

- We continue to work on the share classes (Shape, Circle, Rectangle)

- Add the following,

  - Add a default constructor in each of the 3 classes

    - In each constructor, write a statement to print out something like,

      - "In the default constructor of _____ class." (fill the blank with right class name)

  - Add the instance variable radius to the Circle class, and width and length to the Rectangle class

  - Add parameterized constructors in the Circle and Rectangle class.

    - Initialize the instance variables from the parameters

    - Write a statement to print out something like, "In the constructor _____ of _____ class".

  - Revise the ShapeClient to call the parameterized constructors instead.

  - Make sure your program compiles and runs

Defining a Subclass

- A subclass inherits from a superclass.

- One can also:

  - Add new properties

  - Add new methods

  - Override the methods of the superclass

Overriding Methods in Superclass

- Modify the implementation of a method defined in the superclass

```
public class Circle extends GeometricObject {

  // Other methods are omitted


  /** Override the toString method defined in GeometricObject */
  public String toString() {
    return super.toString() + "\nradius is " + radius;
  }
}
```

Invoking Superclass's Instance Method

- Example

    - One could rewrite the printCircle() method in the Circle class as follows:

        public void printCircle() {

        System.out.println("The circle is created " +

        <u>super</u>.getDateCreated() + " and the radius is " + radius);

        }

Discussion: Method Overriding

- Can you override a private method in the superclass?

Discussion: Method Overriding

- Can you override a private method in the superclass?

  - No

- An instance method can be overridden only if it is accessible.

- A private method is not accessible outside its own class.

- A private method in the superclass can only be accessible in the superclass itself, is inaccessible in the subclass.

- Thus a private method cannot be overridden.

Discussion: Unrelated Methods

- Can you have a method whose <u>signature</u> is identical to a private method in the superclass?

Discussion: Unrelated Methods

- Can you have a method whose <u>signature</u> is identical to a <u>private</u> method in the superclass?

  - Yes

- However, this isn't method overriding. The two methods are unrelated, but happen to have the identical name.

Discussion: Static Method

- Like an instance method, a static method can be inherited.

- However, a static method cannot be overridden.

- If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

Overriding vs. Overloading

- Overriding is to redefine the method with the identical signature in the superclass

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

Two methods with identical name but different signature

Questions?

- Defining subclasses

- A few topics

  - Invoking superclass's methods (constructors and instance methods)

  - Overriding

  - Overriding and overloading

Exercise (Part 3 of 3)

- We continue to work on the share classes (Shape, Circle, Rectangle)

- Add the following,

  - Add a getArea():double method to the Circle and Rectangle class

  - Override getName():String method in the Circle and Rectangle class to include the instance variables and their values, e.g.,, returning something like,

    - Rectangle[width="10.0", length="5.0"]

  - In the ShapeClient class, make you called getName() and getArea() methods on each Circle and Rectangle object you create

  - Make sure your program compiles and runs

(Optional) Exercise

- Listings 11.1 - 11.3 in the textbook define 3 classes: GeometricObject, Circle, and Rectangle.

- In this exercise you are to add two classes to the hierarchy, Triangle and EquilateralTriangle, and write a client class to use the Triangle and EquilateralTriangle classes.

  - The Triangle class is a subclass to GeometricObject, and the EquilateralTriangle isa subclass to Triangle. An EquilateralTriangle is a triangles whose sides are equal.

  - Your program should include 6 files (6 classes): GeometricObject.java, Circle.java, Rectangle.java, Triangle.java, EquilateralTriangle, and TriangleClient.java