CISC 3120

C08: Inheritance and Polymorphism

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap and issues
    - Project progress?
    - Practice assignments?
    - CodeLab?
    - Review guide?
- Inheritance
- Polymorphism via inheritance
- Type casting
- Assignments

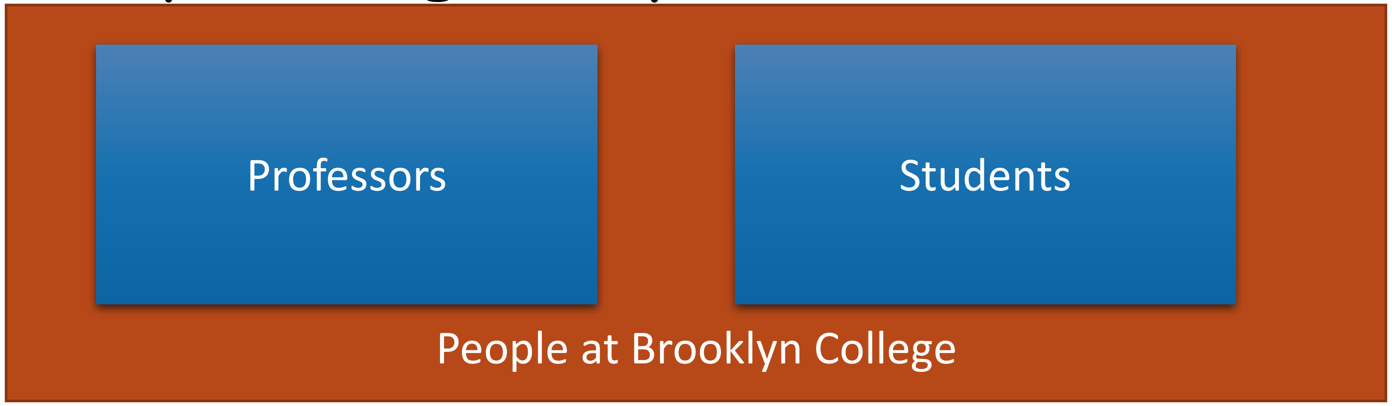
Class and Type

- A class defines a type, and often models a set of entities
- Build a system for managing Brooklyn College, we consider
    - People, a set of individuals (objects), modeled as a class that defines the set of objects

People at Brooklyn College

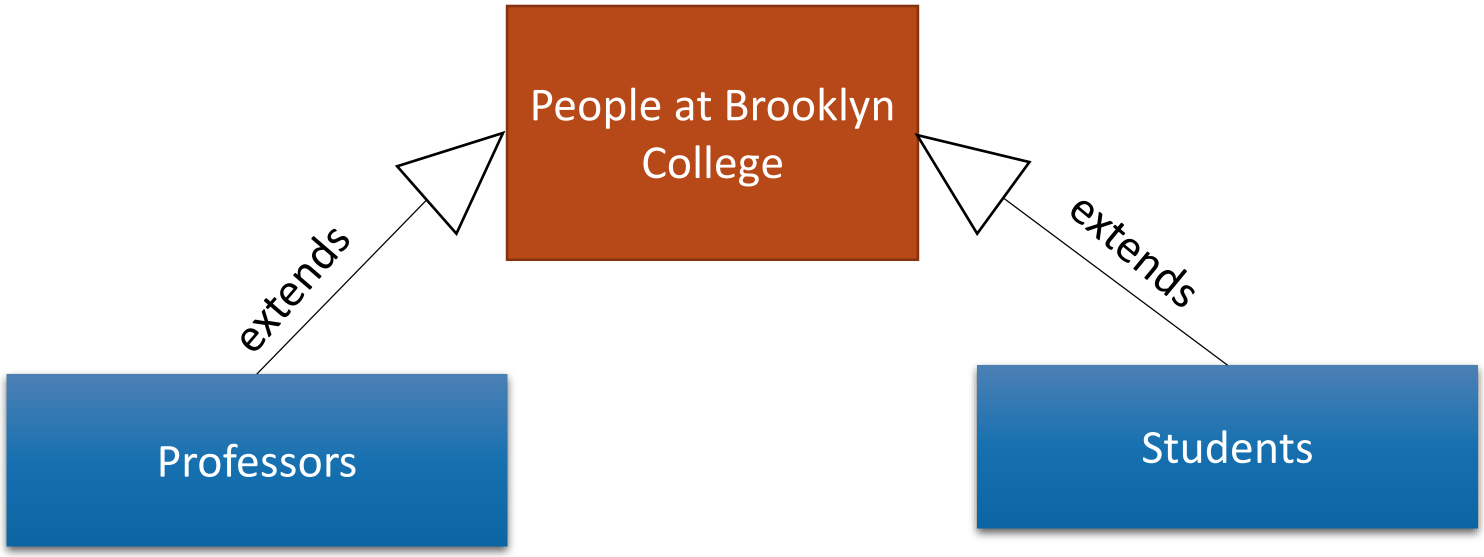
Subtypes

- Some people at Brooklyn are different from the others in some way
- Professors and students are subtypes of Brooklyn College People



Type Hierarchy

- Characteristics and behavior
  - What are Students and Professors in common?
  - What are Students and Professors different?



What’s in common?

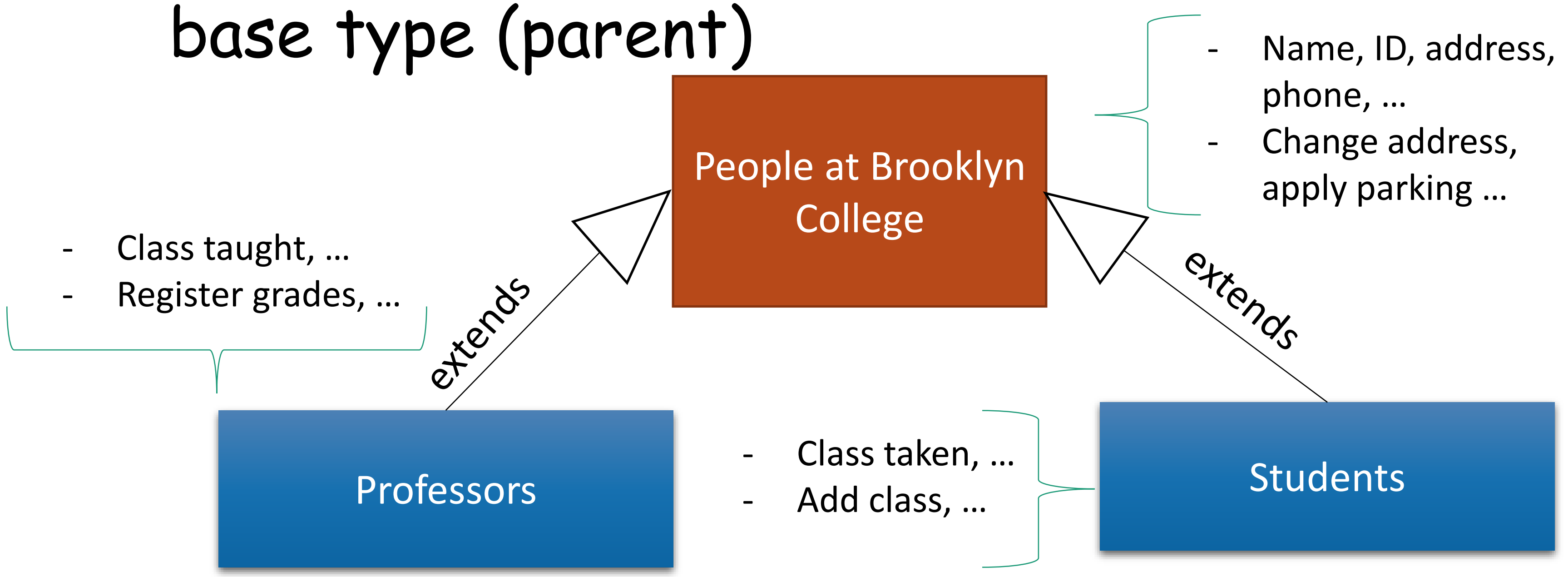
- What characteristics (attributes) and behavior (actions) do People at Brooklyn College have in common?
  - Characteristics (attributes): name, ID, address, email, phone, …
  - Behavior (actions): change address, apply parking, …

What’s Special?

- What’s distinct about students?
  - Characteristics (attributes): classes taken, tuition and fees, …
  - Behavior (actions): add class, drop class, pay tuition, …
- What’s distinct about professors?
  - Characteristics (attributes): course taught, rank, title, …
  - Behavior (actions): register grade, apply promotion, …

Inheritance & Type Hierarchy

- A subtype (child) inherits characteristics (attributes) and behavior (actions) of its base type (parent)



Questions

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

Super Type (Super Class): Person

```
public class Person {
        protected String name;
        protected String id;
        protected 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 Student extends Person {
        private ArrayList<String> classesTaken;
        public Student(String name, String id, String address) {
                super(name, id, address);
                classesTaken = new ArrayList<String>();
        }
        public void haveTakenClass(String className) { …
        }
        public void showClassesTaken() { …
        }
…}
```

Subtype (Subclass): Professor

```
public class Professor extends Person {
        private final static int SABATTICAL_LEAVE_INTERVAL = 7;
        private int yearStarted;
        public Professor(String name, String id, String address, int yearStarted) {
                super(name, id, address);
                this.yearStarted = yearStarted;
        }
        public void applySabbatical(int applicationYear) { …
        }
…}
```

Control Access to Members

```
… protected String name; …
```

Modifier	Class	Package	Subclass	World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
(no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

Choose Access Control Level

- Goal: you want to reduce the chances your class is being misused. Access levels is to help achieve it.
  - Use private unless you have a good reason not to.
  - Use the most restrictive access level that makes sense for a particular member.
  - Avoid public fields except for constants. (Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.)

Constructors

- Initialize attributes of an object when it is being created (or instantiated)
- Subclass's constructor
  - Java will call the parent class's **default** constructor if you do not call **one** of parent's constructors explicitly.
    - You may explicitly call it via "super(…)".

```
… super(name, id, address); …
```

Override Methods in Super Class: Methods

```
public class Person { ...
    public String toString() {
        return "Person (name=" + name + ", id=" + id + ", address=" + address + ")";
    } ...
}
```

```
public class Student extends Person { ...
    public String toString() {
            return "Student (name=" + name + ", id=" + id + ", address=" + address
                    + ", coursesTaken=[" + String.join(", ", classesTaken) + "])";
    } ...
}
```

Override Methods in Super Class: Example

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");

Student adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");

System.out.println (ben.toString());

System.out.println(adam.toString());
```

```
Person (name=Ben Franklin, id=00124, address=2901 Bedford Ave)

Student (name=Adam Smith, id=00248, address=2902 Bedford Ave,
coursesTaken=[])
```

Questions

- Inheritance in Java
- Access control of class members
- Constructors
- Overriding methods
- A few other related items
    - this, super

Polymorphism

- One type appears as and is used like another type
- Example
  - A Student object can be used in place of a Person object.
- Inheritance is an approach to realize polymorphism

Polymorphism: Example 1

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");

Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");

System.out.println (ben.toString());

System.out.println(adam.toString());
```

```
Person (name=Ben Franklin, id=00124, address=2901 Bedford Ave)

Student (name=Adam Smith, id=00248, address=2902 Bedford Ave,
coursesTaken=[])
```

Polymorphism: Example 2

```
public static void display(Person person) {
    System.out.println(person.toString());
}
```

```
    Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");
    Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
    display(ben); display(adam);
```

```
    Person (name=Ben Franklin, id=00124, address=2901 Bedford Ave)
    Student (name=Adam Smith, id=00248, address=2902 Bedford Ave,
    coursesTaken=[])
```

How about Other Methods?

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");
Student adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
adam.haveTakenClass("CISC3120");
display(ben); display(adam);
```

```
Person (name=Ben Franklin, id=00124, address=2901 Bedford Ave)
Student (name=Adam Smith, id=00248, address=2902 Bedford Ave,
coursesTaken=[CISC3120])
```

How about this example?

- You say, "adam" appears to be a "Student" object.

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");
Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
adam.haveTakenClass("CISC3120");
display(ben); display(adam);
```

```
Error: The method haveTakenClass(String) is undefined for the type Person
```

Type Casting

- You can only invoke the method of declared type, i.e., Person.

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");
Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
((Student)adam).haveTakenClass("CISC3120");
display(ben); display(adam);
```

```
Person (name=Ben Franklin, id=00124, address=2901 Bedford Ave)
Student (name=Adam Smith, id=00248, address=2902 Bedford Ave,
coursesTaken=[CISC3120])
```

Actual Type and Declared Type

- Declared type: type at compilation time
- Actual type: type at runtime
  - A variable may refer to an object of different type at runtime
  - Example: actual and declared types of “ben”, and “adam”?

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");

Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");

((Student)adam).haveTakenClass("CISC3120");
```

Type Casting

- Down-casting
  - Cast to a subtype
  - It is allowed when there is a possibility that it succeeds at run time (e.g., type to be casted to matches actual type)
    - In the example, a “Person” object references to a “Student” object, and the down casting is allowed.
- Up-casting
  - Cast to a super type
  - It is always allowed

Questions

- Polymorphism via inheritance in Java
- Type casting in Java

Terms of Choice

- Super type
- Super class
- Base type
- Base class
- Parent class
- Child class
- …

Design Consideration

- Composition vs. Inheritance

More Example: Boat, RowBoat …

- Both examples (Person-Student-Professor and Boat-RowBoat) are in the “sampleprograms” repository on Github

Assignments

- To be available via CUNY Blackboard