CISC 3115 TY2

Polymorphism and Dynamic Binding

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Notice

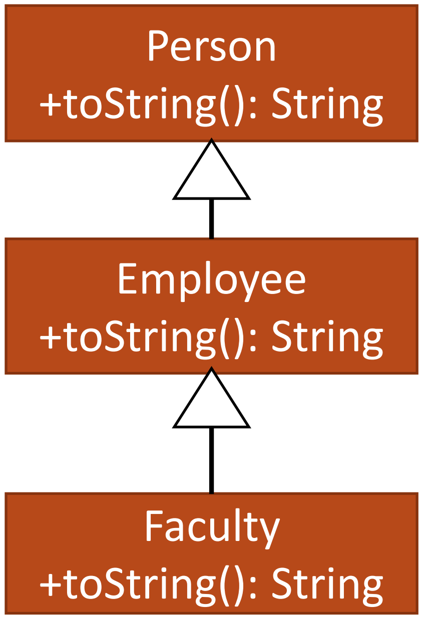
- The slides are subject to change.

Outline

- Discussed
    - Inheritance
        - Superclass/supertype, subclass/subtype
    - Inheritance and constructors in Java; Inheritance and instance methods in Java
    - The Object class in Java
    - Concept of Polymorphism; Polymorphism via inheritance
- Revisiting polymorphism via inheritance
    - What is dynamic binding? How do we design programs with it?
        - Write “generic method” with “generic parameter”
        - How does Java determine the actual type?
- Casting (down-cast and up-cast)
- The instanceof operator

Dynamic Binding

- A method can be implemented in several classes along the inheritance chain. The JVM decides which method to invoke at runtime.



Example: Dynamic Binding

```
public class DynamicBindingDemo {
  public static void main(String[] args) {
    m(new GraduateStudent());
    m(new Student());
    m(new Person());
    m(new Object());
  }

  public static void m(Object x) {
    System.out.println(x.toString());
  }
}
```

```
class GraduateStudent extends Student {
}

class Student extends Person {
  public String toString() {
    return "Student";
  }
}

class Person extends Object {
  public String toString() {
    return "Person";
  }
}
```

Polymorphism Revisited

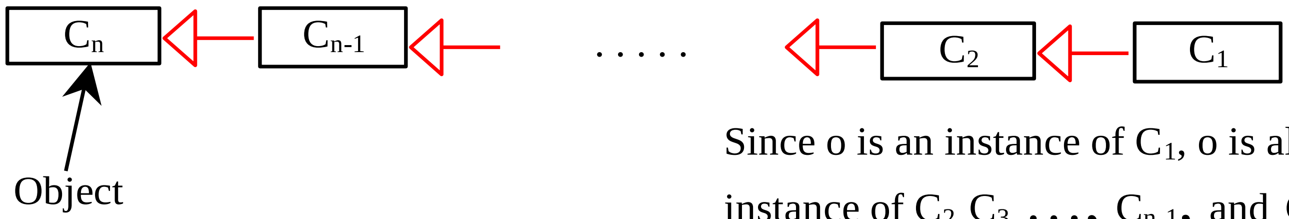
- Take a look at the example

```
public static void m(Object x) {
  System.out.println(x.toString());
 }
```

- Method m takes a parameter of the Object type. You can invoke it with any object.
    - Question: Why?
- An object of a subtype can be used wherever its supertype value is required. This is in effect polymorphism.

Dynamic Binding: General Idea

- A class hierarchy: C_1, C_2, ..., C_{n-1}, and C_n
    - C_n is the most general class, and C_1 is the most specific class.
- Given o of declared type C_i, and of actual type C_j, where i >= j (C_i is more generic than C_j)
    - What happens when o.m()?
- JVM searches the hierarchy for method m
    - the JVM searches the implementation for the method m in C_j, C_{j+1}, ..., C_{i-1} and C_i, in this order, until it is found. Once an implementation is found, the search stops and the first-found implementation is invoked.



Since o is an instance of C_1, o is also an instance of C_2, C_3, . . . , C_{n-1}, and C_n

Method Matching and Binding

- Matching a method signature
  - At <u>compilation time</u>, the <u>Java compiler</u> finds a matching method according to method signature (method name, parameter type, number of parameters, and order of the parameters)
- Binding a method implementation
  - At <u>runtime</u>, the <u>JVM</u> dynamically binds the implementation of the method at runtime since the method may be implemented in any classes in a class hierarchy

“Generic” Programming via Dynamic Binding

- Polymorphism allows methods to be used generically for a wide range of object arguments.
- How?
  - A parameter of the method is of a superclass (declared type)
  - For the parameter, one may pass an object of any of the parameter’s subclasses
  - JVM binds dynamically the implementation of the methods of the object

```
public class PolymorphismDemo {
  public static void main(String[] args) {
    m(new GraduateStudent());
    m(new Student());
    m(new Person());
    m(new Object());
  }

  public static void m(Object x) {
    System.out.println(x.toString());
  }
}

class GraduateStudent extends Student {
}

class Student extends Person {
  public String toString() {
    return "Student";
  }
}

class Person extends Object {
  public String toString() {
    return "Person";
  }
}
```

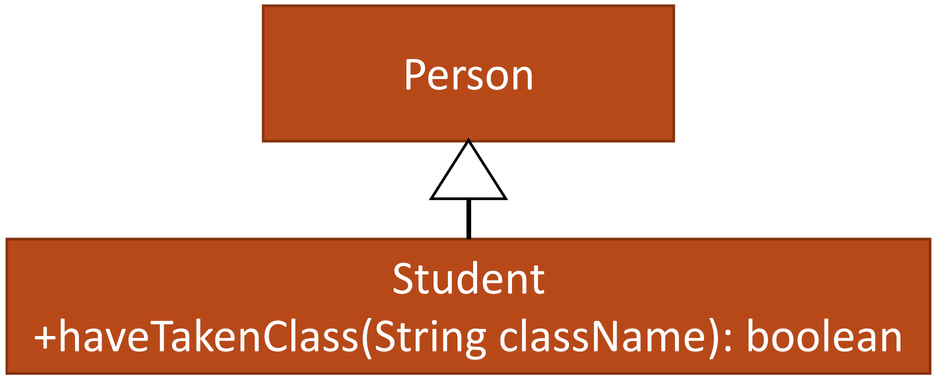
Questions?

- Concept of dynamic binding
- Difference between method matching and binding
- Generic programming via polymorphism and dynamic binding

Casting Objects

- *Casting* can also be used to convert an reference variable of one class type to another within an inheritance hierarchy.
- Down-casting
- Up-casting

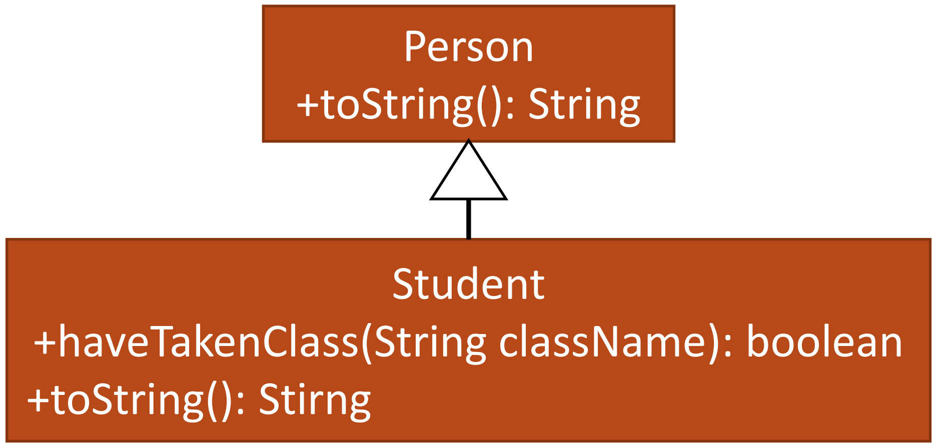
Down-casting



- Cast to a subtype
- It is allowed <u>only</u> when there is a possibility that it succeeds at run time (e.g., type to be casted to matches actual type)
  - Example: Person is a subclass of Student. 1) What is adam and ben's actual type and declared type? 2) Which one is allowed and which isn't? 3) When it isn't, there will be an error. When does the error occurs?

```
Person ben = new Person("Ben Franklin", "00124", "2901 Bedford Ave");
Person adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
((Student)adam).haveTakenClass("CISC3115");
((Student)ben).haveTakenClass("CISC3115");
```

Up-casting



- Cast to a super type
  - It is always allowed

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

- 1) In the above, which method does JVM bind toString() to?
- 2) Is there anything wrong in the below?

```
Student adam = new Student("Adam Smith", "00248", "2902 Bedford Ave");
((Person)adm).haveTakenClass("CISC3115");
```

Implicit Up-casting

- Since up-casting is always allowed, it can be done implicitly
- Example 1
  - Object o = new Student();   // implicit casting
  - Object o = (Object)(new Student());   // explicit casting
- Example 2
  - Assume we have a method
    - void m(Object x) {…}
  - Then
    - m(new Student());   // implicit casting
    - m((Object)(new Student());   // explicit casting
    - Object o = new Student(); m(o);   // implicit casting
    - Object o = new Student; m((Object)o); // explicit casting

Questions?

- Casting
  - Concept
  - Down-casting and explicit casting
  - Up-casting and implicit casting
  - How are they related to declared type and actual type?
  - Under what condition you can do down-casting? Up-casting?

The instanceof Operator

- Dynamic binding may fail at runtime. Programmers should proactively determine whether it succeeds or fails.
- How?
- The instanceof Operator
    - Test whether an object is an instance of a class

The instanceof Operator: Example

```
Object myObject = new Circle();
... // Some lines of code
/** Perform casting if myObject is an instance of Circle */
if (myObject instanceof Circle) {
  System.out.println("The circle diameter is " +
    ((Circle)myObject).getDiameter());
  ...
}
```

$\left.\right\}$ This may fail at runtime. How did we prevent it?

Examples: Dynamic Binding and Casting

- Demonstrating polymorphism, dynamic binding, and casting
- Two examples

Questions?

- Concept of dynamic binding
- Concept of casting
- Concept of declared type and actual type
- The instanceof Operator
- Examples for polymorphism and dynamic binding

Exercise

- Use the class hierarchy of a few fruits in Question 11.9.3 to complete the following tasks
  - Create a subdirectory/folder in today's journal
  - Implement the <u>5 classes</u> with a "<u>name</u>" data field and "<u>toString(): String</u>" method. The return value of the toString() method must contain the class name, and the value of the "name" data field, e.g.,
    - Apple[name="small red"]
  - Add method "getApplePieRecipe(): String" to the Apple class.
  - Add method "getOrangeJuiceRecipe(): String" to the Orange class.
  - Write a FruitClient class where you design a few statements to demonstrate 1) polymorphisms, 2) dynamic binding, and 3) down-casting and up-casting. For each, write a comment explain how the statements you write demonstrate each. Be sure to use instanceof when doing down-casting
  - Submit your journal (6 Java classes)