abstract classes and casting objects

- Professor Evan Korth
- New York University

Road map

- Casting objects
  - instanceof
  - Keyword: abstract
    - Abstract methods
    - Abstract classes

review

- What does it mean to have a protected data member?
- What does Object’s default equals() method test?
- What does Object’s default toString() method return?

Review (cont)

- Given the following class headings:
  public class Ship
  public class FishingBoat extends Ship
  public class SpeedBoat extends Ship

- Which of the following are legal?
  Ship s = new Ship();
  Ship s = new FishingBoat();
  FishingBoat f = new Ship();
  FishingBoat f = new FishingBoat();

Casting objects (review)

- Class variables can refer to objects of related classes as long the object has an “is a” relationship to the class.
- Since subclasses have an “is-a” relationship with their superclass, we can always point a superclass reference variable to a subclass object. In such case a casting operator is not necessary. You implicitly cast the object to it’s superclass just by using the `gets` operator in an assignment statement. For example:
  `Superclass sup = new Subclass();`
  `Or`
  `Subclass subclassVar = new Subclass();
  Superclass sup = subclassVar;`

Casting objects (review)

- On the other hand, an object referenced by a superclass variable may or may not be a subclass object.
- To attempt an assignment such as:
  `Subclass sub = superclassVar;`
  You must explicitly cast the superclass object to the subclass type. For example:
  `Subclass sub = (Subclass) supclVar;`
- Note: If the variable supclVar in the example above is not an instance of Subclass, a runtime error will occur.
Casting objects (cont)

- In order to help prevent the type of runtime error described in the last slide, Java has the keyword `instanceof`, which is used to test a variable to see if it is an instance of a class. For example:
  
  ```java
  (supclVar instanceof Subclass)
  ```

  will evaluate to `true` if and only if `supclVar` references an instance of class `Subclass`.

- So we can test a variable to see if it is an instance of a class before we make an explicit cast. For example:

  ```java
  if (supclVar instanceof Subclass)
  Subclass sub = (Subclass) supclVar;
  ```

  Will never cause a runtime error because the assignment will only happen in the event that `supclVar` is an instance of `Subclass`.

Liang’s example

- If we create a GeometricObject variable, we can “point” it to a Rectangle or a Circle.
  - We can access either Circle’s `toString()` method or Rectangle’s `toString()` method through the reference variable.
  - What if we wanted to access one of their `getArea()` methods?

Abstract Classes and Methods

- Abstract classes
  - Are superclasses (called abstract superclasses)
  - Cannot be instantiated
  - Incomplete
    - subclasses fill in “missing pieces”

- Concrete classes
  - Can be instantiated
  - Implement every method they declare
  - Provide specifics

Abstract Classes and Methods (Cont.)

- Abstract classes not required, but reduce client code dependencies

  - To make a class abstract
    - Declare with keyword `abstract`
    - Contain one or more abstract methods
    - Define abstract methods
      - No implementation, must be overridden

Case Study: Payroll System Using Polymorphism

- Create a payroll program
  - Use abstract methods and polymorphism

  - Problem statement
    - 4 types of employees, paid weekly
      - Salaried (fixed salary, no matter the hours)
      - Hourly (overtime [>40 hours] pays time and a half)
      - Commission (paid percentage of sales)
      - Base-plus-commission (base salary + percentage of sales)
        - Boss wants to raise pay by 10%

Case Study: Payroll System Using Polymorphism

- Superclass `Employee`
  - Abstract method `earnings` (returns pay)
    - Abstract because need to know employee type
    - Cannot calculate for generic employee
  - Other classes extend `Employee`

- Diagram:

  - `Employee`
    - `SalariedEmployee`
    - `CommissionEmployee`
    - `HourlyEmployee`
    - `BasePlusCommissionEmployee`
An abstract method cannot be contained in a nonabstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be declared abstract. In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass.

A subclass can be abstract even if its superclass is concrete. For example, in the book, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract.

You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type. Therefore, the following statement, which creates an array whose elements are of GeometricObject type which is an abstract class, is correct (see book).

```java
GeometricObject[] geo = new GeometricObject[10];
```

Dynamic binding

- The Java Virtual Machine is responsible for binding the correct method to the method call at run time.