Polymorphism

- The term *polymorphism* literally means "having many forms".
- A **polymorphic reference** is a variable that can refer to different types of objects at different points in time.
- The method invoked through a polymorphic reference can change from one invocation to the next.
- All object references in Java are potentially polymorphic.

Polymorphism

- Suppose we create the following reference variable:
  
  ```java
  Occupation job;
  ```

- Java allows this reference to point to an `Occupation` object, or to any object that is-a `Occupation`.
- Remember that an is-a relationship can be established using inheritance or using interfaces.
- Careful use of polymorphic references can lead to elegant, robust software designs.

References and Inheritance

- An object reference can refer to an object of its class, or to an object of any class derived from it by inheritance.
- For example, if the `Holiday` class is used to derive a class called `Christmas`, then a `Holiday` reference could be used to point to a `Christmas` object.

```java
Holiday day;
day = new Christmas();
```
References and Inheritance

- Assigning a child object to a parent reference is considered to be a widening conversion, and can be performed by simple assignment; EX:
- Assigning a parent object to a child reference can be done also, but it is considered a narrowing conversion and must be done with a cast
- The widening conversion is the most useful
- The need for a narrowing conversion between object references often indicates a bad design

Polymorphism via Inheritance

- It is the type of the object being referenced, not the reference type, that determines which method is invoked
- Suppose the Holiday class has a method called celebrate, and the Christmas class overrides it
- Now consider the following invocation:
  ```java
  day.celebrate();
  ```
- If day refers to a Holiday object, it invokes the Holiday version of celebrate; if it refers to a Christmas object, it invokes the Christmas version

Polymorphism via Inheritance

- Now let's look at the Animal class hierarchy from a previous lecture
  - one change: the speakOnce() method is now public (rather than protected)
- The driver Zoo.java illustrates two common uses of polymorphism
  - object method arguments are references, so a method will accept any object that is-a the argument class
  - object array elements are references, so an element can refer to any object that is-a the element class

How Polymorphism Works

- When the compiler sees a method invocation, it verifies that the class of the reference variable has the indicated method

  ```java
  Legged dog = new Dog();
  dog.setAge(5); <- OK
  dog.getBreed(); <- COMPILER ERROR
  ```
- However, the compiler does not determine what code is associated with the method invocation
Binding

* At some point, a method invocation is **bound** to the definition of the method that it invokes
* If this binding occurred at compile time, then that line of code would call the same method every time
* However, Java defers method binding until run time -- this is called **dynamic binding** or **late binding**
* Late binding makes polymorphism possible
  * at run-time, the kind of object on which the method is invoked (and not the class of the reference variable that points to it) determines which version of the method is actually executed

Polymorphism via Interfaces

* An interface name can be used as the type of an object reference variable
* For example, say there exists a `Drawable` interface, with a single abstract method `draw`; we could declare a reference variable:

```java
Drawable d;
```

* The `d` reference can be used to point to any object of any class that implements the `Drawable` interface (i.e., that is-a `Drawable`)
* The version of `draw` that the following line invokes depends on the type of object that `d` is referencing:
  
```java
d.draw();
```

Polymorphism via Interfaces

* Suppose two classes, `Circle` and `Square`, both implement the `Drawable` interface, providing distinct versions of the `draw` method
* In the following code, the first call to `draw` invokes the `Circle` method and the second invokes the `Square` method:

```java
Drawable shape = new Circle();
shape.draw();
shape = new Square();
shape.draw();
```

Polymorphism in Sorting

* Recall that any class that implements the `Comparable` interface defines a `compareTo` method to determine the relative order of its objects
* We can use polymorphism to develop a generic sort for any set of `Comparable` objects
* The sorting method accepts as a parameter an array of `Comparable` objects
* That way, one method can be used to sort a group of CDs, or Accounts, or whatever
* See `PolySort.java` and `SortCD2.java`
  * ignore `Comparable` compiler warnings in Eclipse for now
Lecture 13 Vocabulary

polymorphism
polymorphic reference
binding
dynamic/late binding