Overview

° Problem: Can we delay decisions regarding which method to use until run time?

° Polymorphism
  • Different methods with the same name can be executed for the same reference at different times

° Polymorphism via inheritance
  • References of parent classes can be assigned addresses of child objects

° Flexibility
  • Java was written with the Internet in mind. The changing environment of the web necessitates flexibility
Basic Reference Creation

Cat cat = new Cat();
Animal animal = new Animal();

This involves creating a reference and then assigning an address of an object of the same type

This seems pretty straightforward
More complicated example

- What happens if Cat is a subclass of Animal
  - We can assign the address of Cat to an Animal reference!

- Animal animal = new Cat();

- An object reference of super class type points to an object of sub class type.

- It is legal and it is called polymorphism.

- Since Animal may have several subclasses, multiple types can be assigned to animal
Binding

Consider the following method invocation:

\[ \text{animal.makeNoise();} \]

At some point, this invocation is \textit{bound} to the definition of the method that it invokes.

If binding occurred at compile time, then that line of code would call the same method every time.

Java defers method binding until run time -- this is called \textit{dynamic binding} or \textit{late binding}.

Late binding provides flexibility in program design.
Polymorphism

- The term *polymorphism* literally means "having many forms"

- *A polymorphic reference*
  - Variable that can refer to different types of objects at different points in time

- The method invoked with 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:

  Occupation job;

- Java allows this reference to point to an Occupation object
  - Also could point to any object of a subclass

- This compatibility 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,
  - Also, can refer to an object of any class related to it by inheritance

Example: The Holiday class is used to derive a class called Christmas
  - A Holiday reference (‘day’, in this case) could be used to point to a Christmas object

```java
Holiday day;  // day is a ref to Holiday object
             (Holiday object not yet created)
day = new Christmas();
// Here, day is an object of type Christmas and hence points to Christmas.
```
Assigning a child object to a parent reference is considered to be a widening conversion

- This is what we did in the last slide.
- (a pointer which points to an object of a derived type.)

The opposite: Assigning an parent object to a child reference can be done also,

- It is considered a narrowing conversion and must be done with a cast
- We won’t focus on this in this course

The widening conversion is the more useful
Polymorphism via Inheritance

° In polymorphism:
  • The type of the object being referenced, not the reference type, that determines which method is invoked.

° In previous slides, day is pointing to an object of type Christmas!

° It is very common to have a pointer to a base class pointing to a derived object.
More

° Suppose the `Holiday` class has a method called `celebrate`, and the `Christmas` class overrides it
  • Christmas also has a method called `celebrate`.

° Now consider the following invocation:

  ```java
  day.celebrate();
  ```

° ➔ During execution `day` refers to (points to) a `Holiday` object,
  • It invokes the `Holiday` version of `celebrate`; if it refers to a `Christmas` object, it invokes the `Christmas` version
Create an array of every animal in the Zoo?

Animal animal[] = new Animal[1000];
animal[0] = new Dog();
animal[1] = new Cat();
animal[2] = new Tiger();
animal[3] = new Lion();
......
public void performance(Animal animal)
{
    animal.makeNoise();
}

Consider the following code segment

for (int i = 0; i < 1000; i++)
{
    performance(animal[i]);
}
Consider the following class hierarchy:
Summary

° The basic idea here is that parent references can point to child object

° The specific method which is accessed depends on the type of the child

° This is very flexible since the method used depends on the flow of the program

° Children of abstract classes can also be used in this fashion