CS601: Principles of Software Development

Polymorphism.

Olga A. Karpenko

Parts of this presentations are based on the Java Solftware Solutions book by Lewis&Loftus.

Polymorphism

- Ability to take many forms
- Important object-oriented concept

Polymorphic Reference

- A variable that can refer to different types of objects at different points in time
- All object references are potentially polymorphic

Polymorphic Reference

```
Vehicle v;
```

- Can point to a `Vehicle` object, or to any object of compatible type
- Compatibility can be established using inheritance or using interfaces

References and Inheritance

- An object reference can refer to an object of a subclass

- Example:

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Truck(true); //upcasting
```

Upcasting

- Assigning a child object to a parent reference
- Always fine
- No explicit casting needed
- A widening conversion
- Parent on the left of the assignment

```
Vehicle v = new Truck(true);
```

Parent

Child

Upcasting Examples

- If a Cat is a child of Pet:

```
Pet pet;
Cat cat = new Cat(“Max”, "indoor");
pet = cat;
```

- If Alien is a child of Creature

```
Creature c;
c = new Alien(”Dak-Dak", "Mars");
```

Polymorphism

- The method called through a polymorphic reference can change from one invocation to the next

Example

- Assume Creature is a parent of Alien
- Assume Creature has method speak()
- speak() is overriden in class Alien

```
Creature c1;
c1 = new Creature("blub-blub!");
c1.speak();
c1 = new Alien(”Dak-Dak", "Mars");
c1.speak();
```

- c1 is a polymorphic reference
- two different speak() methods are called
- See `Creature, Alien, Human, Driver`

Binding

- Consider the following method invocation:

```
obj.func();
```

- When is this invocation *bound* to the specific method?
- Does it happen at compile time or run time?

Binding

- Consider the following method invocation:

```
obj.func();
```

- When is this invocation *bound* to the specific method?
- Does it happen at compile time or run time?
  - In Java: runtime (dynamic or late binding)
  - For non-static methods

Polymorphism via Inheritance

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
c1.speak();
```

- What determines which speak() method is invoked?
  - type of the reference variable or
  - the type of object that is referenced

Polymorphism via Inheritance

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
c1.speak(); // child's version
```

- What determines which speak() method is invoked?
  - type of the reference variable or
  - the type of object that is referenced

Polymorphism via Inheritance

- Will anything change?

```
Creature c1 = new Creature("blub-blub!");
c1.speak();
```

Polymorphism via Inheritance

- Will anything change?
- Yes. The parent’s version will be invoked

```
Creature c1 = new Creature("blub-blub!");
c1.speak(); // Creature’s version is called
```

Polymorphism

- Careful use of polymorphic references can lead to elegant, robust software designs

What methods can be called on c1?

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
```

- Public methods of Creature that are not in Alien: Yes.
- Public methods of Creature that are *overridden in Alien*: Yes. Alien’s version will be invoked
- Private methods of Creature: Only if the code is in the Creature class.
- Protected methods of Creature: Only if the code is in a child class

What methods can be called on c1?

- Alien’s methods that are not in Creature: No.
- If `Alien` had a method called `fight()` that `Creature` didn't have:

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
c1.fight();  // compiler error!
```

What methods can be called on c1?

- Alien's methods that are not in Creature: No.
- If `Alien` had a method called `fight()` that `Creature` didn't have:

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
c1.fight();  // compiler error!
```

- A cast can be used to allow the call:

```
((Alien)c1).fight();  // downcasting
```

Downcasting

- Convert a superclass reference to a subclass reference

```
Creature c1 = new Alien(”Dak-Dak", "Mars");
Alien alien = ((Alien)c1); ←—— downcasting
alien.fight();
```

- Need to use explicit cast
- Valid only when superclass reference is actually referencing a subclass object

Casting objects: example

```
Vehicle v1 = new Vehicle();
Truck tr1 = new Truck(true);
```

Casting objects: example

```
Vehicle v1 = new Vehicle();
Truck tr1 = new Truck(true);

Vehicle v2 = tr1;      //upcasting. always ok
```

Casting objects: example

```
Vehicle v1 = new Vehicle();
Truck tr1 = new Truck(true);

Vehicle v2 = tr1;       //upcasting. always ok
Truck tr2 = (Truck)v2; //downcasting. ok
```

Casting objects: example

```
Vehicle v1 = new Vehicle();
Truck tr1 = new Truck(true);

Vehicle v2 = tr1;        //upcasting. always ok
Truck tr2 = (Truck)v2; //downcasting. ok
tr2 = (Truck)v1;         // runtime error!
```

Quick Check

- `Holiday` is the parent of `EasterHoliday` and `ChristmasHoliday`
- Are the following assignments valid?

- `Holiday h = new EasterHoliday();`
- `EasterHoliday e = new Holiday();`
- `ChrismasHoliday c = new EasterHoliday();`
- `EasterHoliday d = (EasterHoliday)h;`

Quick Check

- `Holiday` is the parent of `EasterHoliday` and `ChristmasHoliday`
- Are the following assignments valid?

- `Holiday h = new EasterHoliday();` // yes, upcast.
- ~~`EasterHoliday e = new Holiday()`~~`;`
- ~~`ChrismasHoliday c = new EasterHoliday();`~~
- `EasterHoliday d=(EasterHoliday)h;` // yes, downcast.
  - ok, because h points at an EasterHoliday object
  - Need explicit casting

In-Class Exercise

- Get the starter code for classes Creature, Alien, Human
- Add several Aliens and Humans to the ArrayList
- Iterate over the list
  - call speak() method
  - call fight() method only for Aliens (need to downcast)

Static Methods

- Can **no**t be overriden
- Do **not** behave polymorphically
- Example in Eclipse