CMPE212 – Reminders

• Assn 4 not posted yet. Still…

• Quiz 2 tomorrow in the lab. Topic Coverage in last Wednesday’s lecture. Up to but not including Inheritance.

Today

• Inheritance, Cont.:
  – A coded version of the Person Hierarchy.
  – Another, better version of the OOP Zilch game that eliminates duplicated code.

Back to the Example

• In Java, every new object you define is automatically a child of the Object class.

  So our actual structure is:

  ```
  \begin{tikzpicture}
  \node (O) {Object};
  \node (P) [below of=O] {Person};
  \node (F) [below left of=P] {Professor};
  \node (S) [below right of=P] {Student};
  \draw[->] (O) -- (P);
  \draw[->] (P) -- (F);
  \draw[->] (P) -- (S);
  \end{tikzpicture}
  ```

Example, Cont.

• This is a simplified UML Class Diagram. The upward pointing arrow shows the “is a” relationship.

  So:
  – Person is a Object
  – Professor is a Person
  – Student is a Person
  – Professor is a Object
  – Student is a Object

Aside - the Object Class

• Object is the “Universal” parent class for every object in Java – in your code, in the API, everything!

  So, we are inheriting all the public members of this class – whether we want this stuff or not.

  What members are we getting and how do we find out?

extends Keyword

• Creating the Person class:
  ```java
  public class Person {
  }
  ```

  Which is the same as:
  ```java
  public class Person extends Object {
  }
  ```

  But the “extends Object” is always there, so you don’t have to write it.

  Creating the Professor and Student classes:
  ```java
  public class Professor extends Person {
  }
  public class Student extends Person {
  }
  ```
Example, Cont.

- Suppose we wish to store the description of every person in this class (me included!) in one (small) array.
- What array declaration would you use?
- The following example code is greatly simplified. It does not have any error checking, for example.

```java
public class Person {
    private int age;
    private String name;
    private String gender;
    public Person (int a, String n, String g) {
        age = a;
        name = n;
        gender = g;
    }
}
```

Example, Cont.

```java
public class Professor extends Person {
    private int numExamsToCreate;
    public Professor (int a, String n, String g, int e) {
        super(a, n, g);
        numExamsToCreate = e;
    }
}
```

Example, Cont.

```java
public class Student extends Person {
    private int numExamsToWrite;
    public Student (int a, String n, String g, int e) {
        super(a, n, g);
        numExamsToWrite = e;
    }
}
```

Example, Cont.

```java
public class ClassroomDB {
    public static void main (String[] args) {
        Person[] db = new Person[20];
        db[0] = new Student(18, "Fred", "male", 7);
        db[1] = new Student(19, "Melissa", "female", 7);
        db[2] = new Professor(29, "Alan", "male", 1);
    }
}
```

Example, Cont.

```
super Keyword

- Provides a reference to the immediate parent class.
- In the examples above, it was used to call the constructor in the parent class, Person.
- Note that the compiler will force you to invoke super in a child class' constructor, and it must be the first line in the constructor.
```

```
```
Polymorphism, Again

- Pointers of type Person are pointing to objects of type Student and Professor at runtime.
- At runtime the pointers will morph into their actual object types – this is *late binding* or *dynamic binding*.

Methods in Sub-Classes

- You have five ways to have methods in a sub-class:
  - Write a new method (*concrete* or *abstract*).
  - **Inherit** the method – it is available to use inside the sub-class or it can be invoked from an instance of the sub-class.
  - **Override** the method. You must declare the method with exactly the same method declaration line, the same signature, as was used in the super-class.
  - **Overload** the method in the super-class.
  - **Refine** the method, by invoking the super class’s method in an overridden method.

Methods in Sub-Classes

Inheriting Methods

- As we have discussed before, all public methods from the parent class are inherited by the child.
- And, of course, the parent class may have inherited a bunch of methods from it’s parent class – so every public method in the hierarchy above the current class is available.
- They behave just as if they were written in the child class.

Inheriting Methods, Cont.

- Note that an abstract sub-class can simply inherit an abstract method from an abstract super-class or an interface – it does not have to implement the method. In this case the sub-class will have to be abstract, as well.
- Suppose you don’t want all these methods – how can you avoid inheriting some, but not others?

Method Overriding, Example

- The Halloween5 object overrode the `equals` methods of the `Object` class:

  ```java
  public boolean equals (Object obj)
  ```

Method Overriding & Refining

- When a sub-class overrides a method of the super-class, then it is the sub-class version that is invoked when called from an instance of the sub-class.
- Inside the sub-class, if you wish to invoke the super-class version of the overridden method, use the `super` keyword.
- If the overriding sub-class method invokes the super-class version of the same method, then it has **refined** the super-class method, as well as overriding it.
Aside - the Use of the `final` Modifier

- If "final" is included in a class header line, as in:
  ```java
  public final class ClassName {...}
  ```
  - Then the class cannot be extended.
- If you add "final" to a method definition, as in:
  ```java
  public final void methodName () {...}
  ```
  - Then the method cannot be overridden.

Aside - the Effect of "private"

- `private` methods and attributes are not inherited by the sub-class.
- But, you do inherit `public` methods and attributes.
- You can invoke `public` methods inherited from the parent class even when they themselves refer to attributes and methods that are private in the parent class.

Overloading Methods

- If you use the same method name and return type as in the parent class, but a different parameter list in the sub-class, then you are just overloading the method.

Aside - Multiple Inheritance?

- This is when a sub-class extends more than one super-class.
- In our diagrams, a sub-class would point to more than one super-class.
- Java does not support multiple inheritance.
- (C++ allows it.)
- The designers of Java felt that multiple inheritance would lead to structures that are way too complex.
- Java designers have supplied the use of interfaces as a way of getting around this restriction.

Aside - Multiple Inheritance?, Cont.

- A sneaky way to get around the lack of multiple inheritance in Java is to combine two classes into one, by making one of them an Inner Class, and then you extend the outer class...

Code Example

- Before going further, let us look at expanded code for the simple Person/Student/Professor inheritance structure.
- Some features to look for:
  - Use of `ArrayList<T>`
  - Overridden methods
  - Refined methods
  - Use of `instanceof` keyword
  - Use of `super` keyword
  - Use of `getClass()`
  - Use of `abstract` keyword in an abstract class
  - Polymorphism!!
Polymorphism, Cont.

- Where one type can appear as itself, but ends up being used as another type.
- Java is a very strongly typed language! Even so, you can allow one object to appear to be something else.
- In Java polymorphism must be constructed through object extension or interface implementation.

Polymorphism - Late and Early Binding

- Your program must always satisfy early binding for it to compile.
- Late or dynamic binding occurs when the program is running, and only occurs when a variable of a parent-type object is pointing to a child object.
- The compiler does not (and cannot) check the late binding to see if it works. A failure of late binding results in a runtime error, not a compiler error.

Visualization of Hierarchies

- The easiest way is to view the structure is as a UML (Unified Modeling Language) Class Diagram.
- This is what we have been doing, but normally more detail is shown.
- See our Person Hierarchy in a proper UML Class Diagram on the next slide:

ArgoUML

- Eclipse can be configured to generate UML diagrams...
- It is better to use a stand-alone tool, such as ArgoUML, available from:

  http://argouml.tigris.org

- You can generate a diagram from existing code or even generate skeleton code from a diagram.
- Also, see “List of UML tools” in Wikipedia.

Fixing Up the Zilch OOP Version

- From before: One problem with the last OOP version was that there was quite a bit of duplicated code in the classes AIPlayer and HumanPlayer.
- Let’s have a quick look at the previous OOP version.
- Now, this duplicated code can be put in a parent class: Player, and then AIPlayer and HumanPlayer can extend this parent class. Let’s have a look. Note that Player is abstract – why does this have to be the case?
- No other classes have changed.
- A much better structure!
What's Next?

- You have everything you need to work on Assignment 4.
- You are also ready to do Exercise 9.
- Next topic is Generics in Java:
  - Start by looking at a Generic Collection type from the API – `ArrayList<T>`.
  - This collection has been used in demo and assignment testing code already.