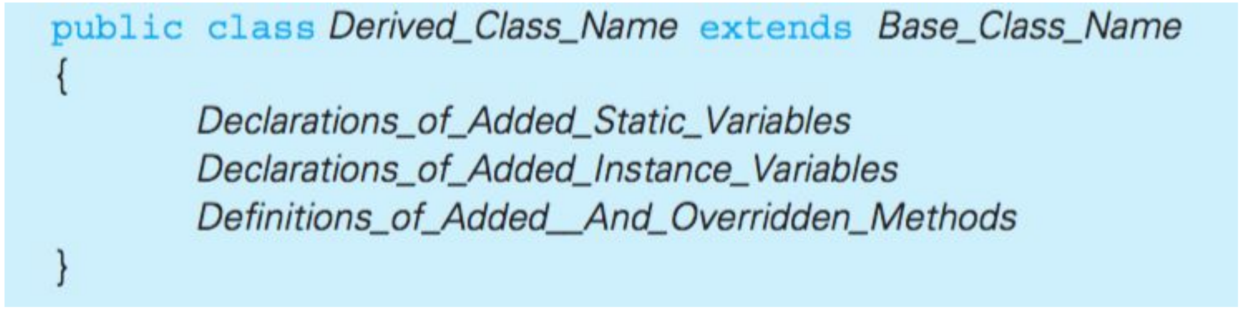
Recitation 8 - February 27th A4/GR

Isaac and William

Inheritance Review

Inheritance Review

- Inheritance: derived class (subclass) inherits all public and protected variables and public methods from base class (superclass)
  - They are not explicitly mentioned in the subclass
  - Subclass has is-a relationship with superclass
- Subclass can also add its own methods and variables on top of what it inherits
- Create a subclass using the keyword extends



```
public class Derived_Class_Name extends Base_Class_Name
{
        Declarations_of_Added_Static_Variables
        Declarations_of_Added_Instance_Variables
        Definitions_of_Added__And_Overridden_Methods
}
```

Constructor Chaining

- “this”: call to another constructor in the same class
- “super”: call to a base class constructor
- Any call to “this” or “super” must be the first action taken
  - In subclass constructor, if no “this” or “super” call, an implicit call to super with no-args is made
  - You can only have one “this” or “super” call in each constructor

```java
public HourlyEmployee()
{
    this("No name", new Date("January", 1, 1000), 0,0);
}
```

```java
public SalariedEmployee(SalariedEmployee originalObject)
{
    super(originalObject); //Invocation of base class
                           //constructor.
    salary = originalObject.salary;
}
```

Polymorphism

What is Polymorphism? (:

Polymorphism refers to “having many forms”

Thus a **polymorphic reference** is a reference variable that can refer to different types of objects at different points in time

E.g. Animal fido = new Dog();

This works because fido is a reference to an Animal, and a Dog instance is-a Animal

So essentially the left side refers to what the reference variable refers to, and the right side refers to what happens during runtime.

So is this possible?

```
Dog d = new Animal()
```

Essentially a Dog reference can only refer to things lower in the hierarchy. Since the entire point of polymorphism is to have the reference be more general than the class that gets instantiated.

As such this doesn’t work.

Compile Errors

1. Assignment. Is the reference type compatible with the object being instantiated?

2. Method calls. Does the reference type (not the object type!) have the method?

3. Casting. Is the reference type in the same hierarchy as the casting type? Does the reference point to something that could potentially be cast to the casting type?

Compile-time assignment errors

```
Animal a = new Dog();
```

Reference/Static Type - Animal
Object/Dynamic Type - Dog

Check if the reference type is higher or equal in the hierarchy than the object

Compile-time method call errors

```
Animal a = new Dog();
a.bark(); // Assuming that animal doesn’t have a bark method
```

To check for compile errors in method calls, look at the reference type and see whether the method exists in that class (or is inherited from an ancestor class). If it doesn't, you have a compile error even if the object has the method.

Consider the following

Why does this not work?

```
int choice =
scan.nextInt();

Animal anim = null;
if (choice == 1) {
anim = new Dog();
} else {
anim = new Cat();
}
anim.bark();
```

Silencing the error

```
Animal anim = null;
if (choice == 1) {
anim = new Dog();
} else {
anim = new Cat();
}
((Dog)anim).bark();

DON’T DO:
(Dog)anim.bark();
```

Compile-time casting errors

```
Dog d = new Dog();

Cat c = (Cat) d; // Why can the compiler flag this?
```

Runtime Errors: ClassCastException

```
Animal a = new Animal();

Dog d = (Dog) a;



Animal b = new Dog();

Cat c = (Cat) b;
```

Dynamic Binding

```
Dog d1 = new Dog();
d1.bark(); // dog bark
Puppy p1 = new Puppy();
p1.bark(); // puppy bark
Dog d2 = new Puppy();
d2.bark(); // which method does this call? Why?
((Dog)d2).bark();  // How ‘bout this one?
```

Abstract

```
public abstract class Animal {

    public abstract void eat();

}
```

Static and Final With Abstract

- Can we have a final abstract method?

- What about a static abstract method?

Overriding vs. Overloading

- Overriding: redefining a method in the subclass that it inherited from the base class
    - Must have exact same name, number, types of parameters, and return type
    - If want to call base class' method and you've overridden it, use super.method()
- If add "final" modifier to a method, you cannot override the method in a subclass
- Overloading: method of same name in derived class has different number or types of parameters
    - Derived class would have access to both methods

Example

Parent Class

```java
public void setName(String firstName, String lastName)
{
   if ( (firstName == null) || (lastName == null) )
   {
      System.out.println("Fatal Error setting employee name.");
      System.exit(0);
   }
   else
      name = firstName + " " + lastName;
}
```

Child Class

```java
public void setName(String newName)
{
   if (newName == null)
   {
      System.out.println("Fatal Error setting employee name.");
      System.exit(0);
   }
   else
      name = newName;
}
```