Class Design

Inheriting Members

Inheriting Methods

- subclass can **override** an inherited method
  - subclass declares a **new implementation** for an inherited method
  - with **same signature** (name & parameters)
  - and **covariant return type**
- the property of the object to take many different forms is called **polymorphism**

```
class Mammal {
  public void speak() {
    System.out.println("Mammal is making a sound.");
  }
}
public class Dog extends Mammal {
  @Override optional, but very useful for avoiding mistakes
  public void speak() { same signature, same return type
    System.out.println("Woof!");
  }
  public static void main(String[] args) {
    Mammal mammal = new Mammal();
    Dog dog = new Dog();
    mammal.speak();
    dog.speak();
  }
}
```

```
Mammal is making a sound.
Woof!
```

```java
// calling method with "super" keyword
class Mammal {
  public void speak() {
    System.out.println("Mammal is making a sound.");
  }
}
public class Dog extends Mammal {
  @Override
  public void speak() {
    System.out.println("Woof!");
    super.speak();
  }
  public static void main(String[] args) {
    Dog dog = new Dog();
    dog.speak();
  }
}
```

```
Woof!

Mammal is making a sound.
```

Method Overriding Rules

1. Overriden method must have **the same signature** as superclass method

2. Overriden method must be at least **as accessible** as the original method

3. Overriden method may not declare a checked exception that is
   new or broader then the one in the original method

4. Return type of overriden method must be **the same or a subtype**
   of the return type of the original method (*covariant return types*)

```java
// covariant return types
class Item {
  protected Number calculatePrice (float price) {
    return price + price * 0.2;
  }
}

public class Shoe extends Item {
  @Override     Double is subtype of Number
  public Double calculatePrice (float shoePrice) {
    return (shoePrice + shoePrice * 0.2) * 1.05;the same signature
  }
  public static void main(String[] args) {
    System.out.println(new Item().calculatePrice(100));
    System.out.println(new Shoe().calculatePrice(100));
  }
}
```

```
120.0
126.0
```

```
// exceptions
// checked exception FileNotFoundException is subclass of IOException
class A {
  public void greet() throws IOException { }
  public void sayHello() { }
  public void leave() {} throws FileNotFoundException {}
}


public class B extends A {
  public void greet() throws FileNotFoundException { }  OK
  public void sayHello() throws IOException { }  NOK
  public void leave() throws IOException { }  NOK
}
```

Overriding private and static methods

- if the method is `private`, it's not visible to other classes
  - the method with the same signature is subclass is independent of that method
  - this is not overriding, it's just completely different method
- it the method is `static`, "overriden" method must also be declared `static`
  - this is not overriding, since every method belongs to its own class
  - this is called *hiding* the method
- methods marked as `final` cannot be overriden nor hidden !!

```java
// hiding a static method
class A {
  public static void greet() { System.out.println("Hello."); }
}

public class B extends A {
  public static void greet() { System.out.println("Good afternoon."); }
  public static void main(String[] args) {
    A.greet();
    B.greet();
  }
}
```

putting @Override here would result with compilation error!

```
Hello.
Good afternoon.
```

```
// variables cannot be overriden, only hidden

class Mammal {

  public String name = "Unknown";

}

public class Dog extends Mammal {

  public String name = "Rex";

  public static void main(String[] args) {

    Dog d = new Dog();

    Mammal m = d;

    System.out.println(d.name);

    System.out.println(m.name);

  }

}
```

Dog's name "hides" Mammal's name

the reference is of the type Mammal, pointing to Dog object

```
Rex

Unknown
```