Subclasses inherit from their superclasses

Declaring a subclass:
```
public class Animal {
  private int weight;
  public void setWeight(int w) { ... };
  public void weight() { return weight;}
  ...
}
```
```
public class Dog extends Animal {
  private int barkVolume;
  public void bark(){ ... }
}
```

Subclasses inherit:
- Data
- Methods

Private data and methods can’t be directly accessed from subclass methods

The is-a relationship

A subclass is-a superclass
- A polygon is-a shape
- An hourly employee is-a employee
- A car is-a vehicle
- A dog is-a animal

For instance variables, we use the has-a relationship
- A car has-a wheel
- A dog has-a bark volume

Any operations you can do to a superclass, you can do to the subclass as well.
- The operations may work differently, but they’re still meaningful
Overriding methods

Inherit:
  - State
  - Behavior

Add new:
  - State
  - Behavior

Modify:
  - Behavior
  - Override the method by redeclaring it

Overridden methods:
  - May want to completely replace the old method
  - May want to augment the old method’s behavior

All classes inherit from Object

Either
  - directly (no extends)
  - indirectly

Object defines several methods
  - String toString()
    - By default, returns getClass().getName() + '@' + Integer.toHexString(hashCode())
  - Boolean equals(Object obj)
    - By default, only if obj == this
  - Class getClass()
    - Returns the class of this object
  - int hashCode()
    - Returns a number: if a.equals(b), then their hash codes are equal

The equals method

Must be an equivalence relation
  - reflexive
    - a.equals(a) must be true
  - transitive
    - if a.equals(b) and b.equals(c), then a.equals(c)
  - symmetric
    - if a.equals(b) then b.equals(a)
  - doesn’t equal null
    - except that null.equals(null) is true

The following won’t work

```java
public class Dog extends Animal {
    private int barkVolume;

    public boolean equals(Object o) {
        if (o == null)
            return false;
        else if (!o instanceof Dog)
            return false;
        else {
            Dog d = (Dog) o;
            return this.barkVolume == d.barkVolume;
        }
    }
}
```
The equals method

The following will almost work

```java
public class Dog extends Animal {
  private int barkVolume;
  public boolean equals(Object o) {
    if (o == null)
      return false;
    else if (getClass() != o.getClass())
      return false;
    else {
      Dog d = (Dog) o;
      return this.barkVolume == d.barkVolume;
    }
  }
}
```

Here's what Animal's equals looks like:

```java
public class Animal {
  private int weight;
  public boolean equals(Object o) {
    if (o == null)
      return false;
    else if (getClass() != o.getClass())
      return false;
    else {
      Animal a = (Animal) o;
      return this.weight == a.weight;
    }
  }
}
```

The equals method

The following will work

```java
public class Dog extends Animal {
  private int barkVolume;
  public boolean equals(Object o) {
    if (o == null)
      return false;
    else if (getClass() != o.getClass())
      return false;
    else {
      Dog d = (Dog) o;
      return super.equals(o) &&
        this.barkVolume == d.barkVolume;
    }
  }
}
```

How come assertEquals didn't work for Rationals?

```java
public class Rational {
  ...
  public boolean equals(Rational r) {
    return numerator==r.numerator &&
    denominator == r.denominator;
  }
}
```

```java
public class TestCase {
  ...
  public void assertEquals(Object o1, Object o2){
    assertTrue(o1.equals(o2));
  }
}
```
Using super and this in constructors

When you subclass a class, the constructor for the subclass must make sure all instance variables are initialized.

- If you don’t specify otherwise, the default constructor will be called automatically as first line of the constructor.
- You can specify explicitly:
  ```java
  - public Dog(int volume) {
    super();
    barkVolume = volume;
  }
  - public Poodle(int volume, int curly) {
    super(volume);
    curliness = curly
  }
  - The call to the super() constructor must be the first line of the constructor
- You can also call another constructor in the same class with this:
  ```java
  `public Dog() { this(DEFAULT_BARK_VOLUME); }`
  ```

Using super in other methods

```java
public class Animal {
  int weight;
  public Animal(int w) { weight = w; }
  public Animal(Animal a) { weight = a.weight; }
  String toString() {
    return super.toString() + " weight: " + weight;
  }
}
```

```java
public class Dog extends Animal {
  int barkVolume;
  public Dog(int w, int v) {
    super(w);
    barkVolume = v;
  }
  public Dog(Dog d) { super(d); barkVolume = d.barkVolume; }
  public toString() {
    return super.toString() + " barkVolume: " + barkVolume;
  }
}
```

```java
public class Poodle extends Dog {
  int curliness;
  public Poodle(int w, int v, int c) {
    super(w, v);
    curliness = c;
  }
  public Poodle(Poodle p) { super(p); curliness = p.curliness; }
  public toString() {
    return super.toString() + " curliness: " + curliness;
  }
}
```

Substitution

A variable that can hold a reference to an object of type A can also hold a reference to an object of any subclass of type A

- Given
  ```java
  Employee e = new Employee();
  SalariedEmployee s = new SalariedEmployee();
  HourlyEmployee h = new HourlyEmployee();
  public void addEmployee(Employee e)
  ```
- All the following are allowed:
  ```java
  addEmployee(e);
  addEmployee(s);
  addEmployee(h);
  ```
- Not OK:
  ```java
  s = e;
  s = h;
  ```

Programming Assignment 3

Very similar to project 6 on page 473

- You’ll be creating three classes:
  ```java
  - Person
  - Vehicle
  - Truck
  ```
- You’ll be creating three test cases:
  ```java
  - TestPerson
  - TestVehicle
  - TestTruck
  ```
- We’ll be testing your classes with our unit tests
- We’ll be testing your unit tests with our classes
  ```java
  - Our classes may very well have errors; your unit tests should catch them!
  ```
Programming Assignment 2 debrief

```java
public String toString() {
    Rational i = imaginaryPart();
    String connector = " + i";
    if (i.numerator() < 0) {
        i = i.negate();
        connector = " - i";
    } else if (i.numerator() == 0) {
        return realPart().toString();
    }
    return realPart().toString() + connector + i.toString();
}
```

```java
public static void testToStringIntegersImaginary() {
    Complex c = new Complex(1, -2);
    assertEquals("1 - i2", c.toString());
}
```

```java
public static void testToStringRationals() {
    Complex c = new Complex(oneHalf, threeQuarters);
    assertEquals("1/2 + i3/4", c.toString());
}
```

```java
public static void testToStringMissingImaginary() {
    Complex c = new Complex(oneHalf, zero);
    assertEquals("1/2", c.toString());
}
```

```java
public static void testDivisionByZero() {
    Complex c = new Complex(2, 3);
    Rational zero = new Rational();
    try {
        c = c.quotient(zero);
    } catch (ArithmeticException e) {
        fail("should raise an ArithmeticException (divide-by-zero)");
    }
    catch (ArithmeticException e) {
        assertTrue(true);
    }
}
```

```java
public static void testConjugate() {
    Complex c = new Complex(3, 2);
    assertComplexEquals(3, -2, c.conjugate());
}
```

```java
public static void testModulus() {
    Complex c = new Complex(4, -3);
    assertEquals(5.0, c.modulus(), 0.0001);
}
```

```java
public static void testDivisionByZero() {
    Complex c = new Complex(2, 3);
    Rational zero = new Rational();
    try {
        c = c.quotient(zero);
    } catch (ArithmeticException e) {
        fail("should raise an ArithmeticException (divide-by-zero)");
    }
    catch (ArithmeticException e) {
        assertTrue(true);
    }
}
```

```java
public static void testConjugate() {
    Complex c = new Complex(3, 2);
    assertComplexEquals(3, -2, c.conjugate());
}
```

```java
public static void testModulus() {
    Complex c = new Complex(4, -3);
    assertEquals(5.0, c.modulus(), 0.0001);
}
```