Homework Hints: String

Since a String is an object, the equality operator tells us if two String values have the same reference, not whether they contain the same characters.

```java
public static void main(String[] args) {
    String s = "foo";
    String t = "foo";
    boolean b = s==t;
    /* May or may not be true.
     * (Implementation dependent.) *
     * for( int i=0 ; i < args.length ; i++ )
     * if( args[i].equals("--random") )
     *     /* Will probably never be true. */
     *     ...
     * }
```

The equals Method

The String class has a method equals that tests whether two String values represent the same sequence of characters.

```java
public static void main(String[] args) {
    String s = "foo";
    String t = "foo";
    boolean b = s.equals(t);
    /* b=true */
    for( int i=0 ; i < args.length ; i++ )
    if( args[i].equals("--random") )
        /* true if user specified --random. */
        ...
}
```

Other String Methods

String has lots of useful methods that you can look up in the API documentation online. Two more that will come in handy if you plan to do the extra credit:

```java
public boolean startsWith(String prefix) {
    /* determines whether the String starts with the String prefix */
}
```

```java
public String substring(int beginIndex) {
    /* returns a substring beginning at beginIndex and continuing to the end of the String. */
}
```

Review: Objects

An object is a combination of data and operations that manipulate that data. In Java, we call object data “fields” and object operations “methods”.

Let’s try to put everything we know about classes together with an example. We’ll write a simple program to help a corporate personnel department keep track of its employees.

Since employees are the central concern of our program, our core class should probably be Employee.

Employee Code

```java
public class Employee {
    private String name;
    protected double salary;
    public Employee(String name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    public String getName() { return name; }
    public double weeklyPay() {
        return salary / 52.0;
    }
    public void giveRaise(double percent)
    {
        salary += salary * percent;
    }
}
```

Employee Code: cont’d

```java
/* salary represents an annual total, divide by 52 weeks to get weekly pay */
public double weeklyPay() {
    return salary / 52.0;
}
```

Extending a Class

A manager is also an employee, but management has its perks. A manager has a name and a salary, but he also has a secretary. Plus, he gets a bonus.

Since a manager is an employee and also more than an employee, it would be nice if we could somehow extend the Employee class to handle managers. It turns out we can do exactly that.
public class Manager extends Employee {
    private String secretaryName;
    private double bonus;

    public Manager(String name, double salary, String secretaryName) {
        super(name, salary);
        this.secretaryName = secretaryName;
        bonus = 0.0;
    }

    public double weeklyPay() {
        double b = bonus;
        bonus = 0.0;
        return salary/52.0 + b;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        bonus = salary * 0.10;
    }
}

public class Executive extends Manager {
    private double sharesOfStock;

    public Executive(String name, double salary, String secretaryName, double sharesOfStock) {
        super(name, salary, secretaryName);
        this.sharesOfStock = sharesOfStock;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}

public class HourlyEmployee extends Employee {
    private double hoursWorked;
    private double overtimeRate;

    public HourlyEmployee(String name, double salary, double overtimeRate) {
        super(name, salary);
        this.overtimeRate = overtimeRate;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}

public class Executive extends Manager {
    private double sharesOfStock;

    public Executive(String name, double salary, String secretaryName, double sharesOfStock) {
        super(name, salary, secretaryName);
        this.sharesOfStock = sharesOfStock;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}

public class HourlyEmployee extends Employee {
    private double hoursWorked;
    private double overtimeRate;

    public HourlyEmployee(String name, double salary, double overtimeRate) {
        super(name, salary, double sharesOfStock) {
            super(name, salary, secretaryName);
            this.sharesOfStock = sharesOfStock;
        }
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}

public class Manager extends Employee {
    private String secretaryName;
    private double bonus;

    public Manager(String name, double salary, String secretaryName) {
        super(name, salary);
        this.secretaryName = secretaryName;
        bonus = 0.0;
    }

    public double weeklyPay() {
        double b = bonus;
        bonus = 0.0;
        return salary/52.0 + b;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        bonus = salary * 0.10;
    }
}

public class Executive extends Manager {
    private double sharesOfStock;

    public Executive(String name, double salary, String secretaryName, double sharesOfStock) {
        super(name, salary, secretaryName);
        this.sharesOfStock = sharesOfStock;
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}

public class HourlyEmployee extends Employee {
    private double hoursWorked;
    private double overtimeRate;

    public HourlyEmployee(String name, double salary, double overtimeRate) {
        super(name, salary, double sharesOfStock) {
            super(name, salary, secretaryName);
            this.sharesOfStock = sharesOfStock;
        }
    }

    public void weeklyPay() {
        double dividend = sharesOfStock * 0.001;
        return super.weeklyPay() + dividend;
    }

    public void giveRaise(double percent) {
        super.giveRaise(percent);
        sharesOfStock += sharesOfStock * percent;
    }
}
HourlyEmployee Code: cont’d

```java
/* salary now represents the hourly rate */
public double weeklyPay() {
    if (hoursWorked < 40)
        return hoursWorked * salary;
    else {
        double overtime = hoursWorked - 40;
        return 40 * salary + overtime * salary * overtimeRate;
    }
}
```

Class Hierarchies

The relationships between sub- and super-classes form a class hierarchy.

```
class Manager extends Employee {
    public Manager() {
        ... super();
    }
}
```

```
class HourlyEmployee extends Manager {
    public HourlyEmployee(String name, double salary, double overtimeRate) {
        super(name, salary);
        this.overtimeRate = overtimeRate;
    }
}
```

```
class Executive extends Manager {
    public Executive(String name, double salary, String boss) {
        super(name, salary);
        this.boss = boss;
    }
}
```

The Object Class

The `Object` class sits atop the Java class hierarchy. If a class does not explicitly extend another class, it implicitly extends `Object`.

It turns out `toString` is not a “magic” method at all: it is an instance method of `Object` that we can override in our own classes (subclasses of `Object`).

`equals` is also an method on `Object`. The default implementation compares references. `String` overrides `equals` to compare character sequences.

Polymorphism

The assignment-compatibility of a subclass to its superclass types is called `polymorphism`, from the Greek for “many forms”.

An object type declaration can be satisfied by more than one form of a class: it can be satisfied by the class itself, or by any of its subclasses. Java treats the many forms of the class as if they were the same, but we can specialize them by overriding methods.

```
public class Personnel {
    public static void main(String[] args) {
        Employee[] staff = new Employee[5];
        staff[0] = new Employee("Jane Doe", 35000.00);
        staff[1] = new Employee("Mary Jones", 50000.00);
        staff[2] = new Manager("Pointy-Haired Boss", Jane Doe, 70000.00);
        staff[3] = new Executive("H.T. Poindexter III", Mary Jones, 4000000.00, 20000000.00);
        staff[4] = new HourlyEmployee("Joe Blow", 8.00, 1.5);

        System.out.println("Weekly Pay Report:");
        for (int i = 0; i < staff.length; i++) {
            double pay = staff[i].weeklyPay();
            // Calls the appropriate weeklyPay for each subclass
            System.out.println(staff[i].getName() + " : " + pay);
        }
    }
}
```

Constructors in Subclasses

Subclasses do not inherit the superclass’s constructors.

```
public class A {
    public A() { ... }
    public A(int x) { ... }
}
```

```
public class B extends A {
    public B(int x) {
        super(x);
    }
}
```

```
public class Personnel {
    public static void main(String[] args) {
        A a = new A();
        A b = new A(3);
        /* Error: B has no 1-arg constructor */
    }
}
```

```
public class A {
    public A() { ... }
    public A(int x) { ... }
}
```

```
public class B extends A {
    public B(int x) {
        super(x);
    }
}
```

```
public class Personnel {
    public static void main(String[] args) {
        B b = new B(3);
        /* OK: B has a 1-arg constructor */
    }
}
```

Polymorphism: Example

```
public class A {
    public A() { ... }
    public A(int x) { ... }
}
```

```
public class B extends A {
    public B(int x) {
        super(x);
    }
}
```

```
public class Personnel {
    public static void main(String[] args) {
        A a = new A();
        A b = new B(3);
    }
}
```

```
private class A {
    public A() { ... }
    public A(int x) { ... }
}
```

```
private class B extends A {
    public B(int x) {
        super(x);
    }
}
```

```
private class Personnel {
    public static void main(String[] args) {
        A a = new A();
        A b = new B(3);
    }
}
```

Polymorphism: Example, cont’d

```
public class A {
    public A() { ... }
    public A(int x) { ... }
}
```

```
public class B extends A {
    public B(int x) {
        super(x);
    }
}
```

```
public class Personnel {
    public static void main(String[] args) {
        Person staff = new Person[5];
        staff[0] = new Person("Jane Doe", 35000.00);
        staff[1] = new Person("Mary Jones", 50000.00);
        staff[2] = new Person("Pointy-Haired Boss", Jane Doe, 70000.00);
        staff[3] = new Person("H.T. Poindexter III", Mary Jones, 4000000.00, 20000000.00);
        staff[4] = new Person("Joe Blow", 8.00, 1.5);
        System.out.println("Weekly Pay Report:");
        for (int i = 0; i < staff.length; i++) {
            Person p = staff[i];
            System.out.println(p.getName() + " : "+p.salary);
        }
    }
}
```
Constructors in Subclasses: 2

If we don’t call a superclass constructor explicitly, Java will call `super()` implicitly.

```java
public class A {
    public A() { ... }
}

public class B extends A {
    private int x;
    public B(int x) {
        /* Implicit call to A() here */
        this.x = x;
    }
}
```

Constructors in Subclasses: 3

The implicit call to `super()` relies on the existence of a no-arg constructor in the superclass.

```java
public class A {
    public A(int x) { ... }
}

public class B extends A {
    private int y;
    public B(int y) {
        /* Error: no method A() */
        this.y = y;
    }
}
```

Hidden Fields

If you define a field in your subclass with the same name as a field in the superclass, the subclass field hides the superclass’s field.

```java
public class A {
    protected int x;
}

public class B extends A {
    private int x;
    public B() {
        x = 1;
        /* Can’t see A.x from here */
    }
}
```

Hidden Fields: 2

We can access a hidden field of the superclass using `super`.

```java
public class A {
    protected int x;
}

public class B extends A {
    private int x;
    public B() {
        x = 1;
        super.x = 2;
        /* Initializes A.x */
    }
}
```

Narrowing Conversions

You can attempt to cast a superclass object to a subclass type, but it will only work if the object actually is an instance of that subclass.

```java
public class A {
    ... };

public class B extends A {
    public static void main(String[] args) {
        A a1 = new A();
        B b1 = new B();
        A a2 = b1;  /* widening conversion */
        B b2 = (B) a1;  /* Error: a1 is not a B */
        B b3 = (B) a2;  /* OK: a2 is a B */
    }
}
```

The `instanceof` Operator

We can avoid a runtime class cast error by using the `instanceof` operator.

```java
public class A {
    ... };

public class B extends A {
    public static B foo(A a) {
        if (a instanceof B) {
            /* We are now guaranteed this will work */
            return (B) a;
        } else
            return null;
    }
}
```

The `instanceof` Operator: 2

`instanceof` actually tells us if the object is assignment-compatible to the type. They class does not have to match exactly.

```java
public class A { ... };

public class B extends A {
    public static void main(String[] args) {
        B b = new B();
        boolean is = b instanceof A;
        /* b = true */
    }
}
```