

CS100J Final Class on Classes 10 February 2005

Attitude by Charles Swindoll.

The longer I live, the more I realize the impact of **attitude** on life.

Attitude, to me, is more important than facts, the past, education, money, failures, circumstances, success, and what other people think, or say, or do.

It is more important than appearance, giftedness, or skill. It will make or break a company, a church, a home.

The remarkable thing is we have a choice every day regarding the attitude that we will embrace for that day. We cannot change the past. We cannot change the fact that people will act in a certain way. We cannot change the inevitable. The only thing we can change is our attitude.

I am convinced that life is 10% what happens to me and 90% how I react to it. And so it is with you also. We are all in charge of our own **attitude**.

1

Help: Get it now if you need it!!

- One-on-one help from TAs. For info, get on the course website and click "Staff-info".
- Call Cindy Pakkala 255-8240 for an appointment with Gries.
- See a consultant in the eng. library during the day and at Purcell Union in the evening. –they aren't very busy now. Click on "consulting hours" on the home page for the course.
- Peer tutoring (free). On <http://www.engineering.cornell.edu>, click on "student services". On the page that comes up, click on "Learning Initialives (L.I.F.E.)" in the left column, upper part. Then, click on "peer tutoring" in the left column.

2

Content of this lecture

This lecture contains some final miscellaneous points to round out your knowledge of classes and subclasses. There are a few more things to learn after that, but we'll handle them much later.

- Inheriting fields and methods and Overriding methods. [Sec. 4.1 and 4.1.1](#): pp. 142–145
- Function toString. [Sec. 3.1.4](#), pp. 112–113.
- Purpose of **super** and **this**. [Sec. 4.1.1](#), pp. 144–145.
- More than one constructor in a class; another use of **this**. [Sec. 3.1.3](#), pp. 110–112.
- Method equals in class Object. [Sec. 4.3 and 4.3.1](#), pp. 154–155. (We do not cover the method at the end of [Sec. 4.3.1](#).)
- Constructors in a subclass —calling a constructor of the super-class. [Sec. 4.1.3](#), pp. 147–148.

3

```
public class Employee {/** Instance: a person's name, year hired, and salary */
    private String name; // Employee's name
    private int start; // Year hired
    private double salary= 50000; // Salary
    /** Constructor: a person with name n, year hired d, salary s */
    public Employee(String n, int d) { name= n; start= d; salary= s;}
    /** = name of this Employee */
    public String getName() { return name; }
    /** Set the name of this Employee to n */
    public void setName(String n) { name= n; }
    /** = year this Employee was hired */
    public int getStart() { return start; }
    /** Set the year this Employee was hired to y */
    public void setStart(int y) { start= y; }
    /** = Employee's total compensation (here, the salary) */
    public double getCompensation() { return salary; }
    /** Change this Employee's salary to d */
    public void changeSalary(double d) { salary= d; }
    /** = String representation of this Employee */
    public String toString()
        { return getName() + ", year " + getStart() + ", salary " + salary; } }
}
```

This class is on page 105 of the text. It will also be placed on the web site

4

Employee c= new Employee("Gries", 1969, 50000); [Sec. 4.1, page 142](#)
c.toString()

Which method toString() is called?

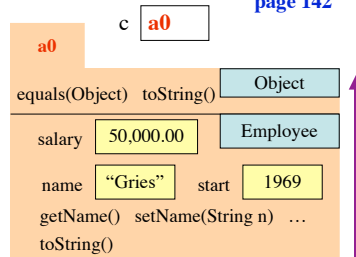
Overriding rule:

To find out which is used, start at the bottom of the class and search upward until a matching one is found.

Also called the **bottom-up rule**.

Terminology. Employee **inherits** methods and fields from Object. Employee **overrides** function toString.

5



Purpose of function toString: to give a string representation of the folder (object) in which it appears. [Sec. 3.1.4, page 112](#)

In Object, all toString can do is to give the name on the folder.

In Employee, toString can tell you the values of the fields

```
/** = String representation of this Employee */
public String toString() {
    return getName() + ", year " + getStart() + ", salary " + salary;
}
```

Nice Java rule. If you use the name c of a folder in a place where a String is needed, Java uses the value of c.toString().

6

Purpose of super and this

Sec. 4.1, pages 144-145

Use **this** to refer to the object in which it appears
 Use **super** to refer to components in the super-class partition of the object (and above).

```
/** = String representation of this Employee */
public String toString() {
    return this.getName() + ", year " + getStart() + ", salary " + salary;
}
```

ok, but unnecessary

```
/** = toString value from superclass */
public String toStringUp() {
    return super.toString();
}
```

necessary

A second constructor in Employee
 Provide flexibility, ease of use, to user

Sec. 3.1.3, page 110

```
/** Constructor: a person with name n, year hired d, salary s */
public Employee(String n, int d, double s)
    { name=n; start= d; salary= s; } First constructor
```

```
/** Constructor: a person with name n, year hired d, salary 50,000 */
public Employee(String n, int d)
    { name= n; start= d; salary= 50000; } Second constructor; salary is always 50,000
```

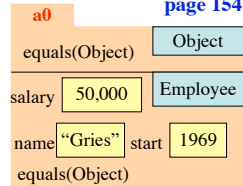
```
/** Constructor: a person with name n, year hired d, salary 50,000 */
public Employee(String n, int d)
    { this(n, d, 50000); } Another version of second constructor; calls first constructor
```

Here, **this** refers to the other constructor

Method equals in class Object.

Sec. 4.3.1, page 154

```
/** = "the name of this object is the same as the name of obj */
public boolean equals(Object obj)
    { return this == obj; }
```



Write equals in class Employee

```
/** = "e is an Employee, with the same fields as this Employee */
public boolean equals(Employee e) {
    return e != null
        && this.name == e.name
        && this.start == e.start
        && this.salary == e.salary;
}
```

Function does not override equals in Object because the parameter has a different type. It's a new, different function. We'll fix redo this function later in the course.

Subclass Executive

```
/** An executive: an employee with a bonus. */
public class Executive extends Employee {
    private double bonus; // yearly bonus
    /** Constructor: name n, year hired d, salary 50,000, bonus b */
    public Executive(String n, int d, double b) {
        super(n, d);
        bonus= b;
    }
    /** = this executive's bonus */
    public double getBonus() { return bonus; }
    /** = this executive's yearly compensation */
    public double getCompensation()
        { return super.getCompensation() + bonus; }
    /** = a representation of this executive */
    public String toString()
        { return super.toString() + ", bonus " + bonus; }
}
```

super. means that the function in the superclass will be called.

Subclass Executive and its folders

Sec. 4.1.3, page 147

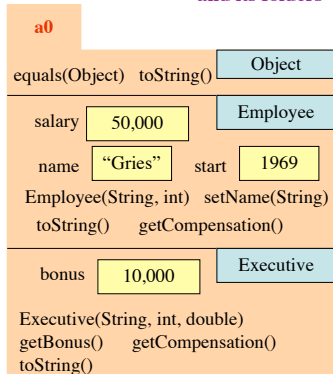
```
public class Executive extends Employee {
    private double bonus;

    public Executive(String n, int d, double b) { ... }

    public double getBonus() { ... }

    public double getCompensation() { ... }

    public String toString() { ... }
}
```



Calling a superclass constructor from the subclass constructor

Sec. 4.1.3, page 147

```
public class Executive extends Employee {
    private double bonus;
    /** Constructor: name n, year hired d, salary 50,000, bonus b */
    public Executive(String n, int d, double b) {
        super(n, d);
        bonus= b;
    }
}
```

The first (and only the first) statement in a constructor can be a call to a constructor of the superclass. If you don't put one in, then this one is automatically used:

super();

Principle: Fill in superclass fields first.

