COMP 110-003
Introduction to Programming
Final Exam Review

April 23, 2013

Haohan Li
TR 11:00 – 12:15, SN 011
Spring 2013

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

General Instructions

- The exam will be more like our midterm rather than the sample exams
  - The exam will take 3 hours. Thus the amount of questions will be basically doubled
  - There won't be optional questions this time (no "choose 2 from 3"). You need to complete all questions. But there are still extra points questions
- Comments are not required. However, you may earn partial credits from them
  - Don't give up – you know the exam values 25%

Computer Basics

- What is --
  - Bit
  - Byte
  - Instruction
  - Program
  - Algorithm
  - Compiler
  - CPU
  - Memory
    - Memory address

Primitive Types

- What are primitive types?
  - int, byte, short, long, float, double, char, boolean
- What are the arithmetic operations
  - Unary operators
    - -, ++, -- (especially, remember "-" can mean "negative")
  - Binary arithmetic operators
    - +, -, *, /, % (recall "mod")
- Parentheses and precedence
  - Parentheses > unary > binary

Primitive Types

- Type casting
  - Implicit converting
    - byte->short->int->long->float->double
    - This can be automatically done
    - **Recall: double d = int1 / int2;**
  - Explicit casting
    - In the other direction
    - You must explicitly write the casting
    - **Recall: int i = (int)(double1 / double2);**

Primitive Types

- Type casting
  - Java casts types only when they don't match
  - Sample question:

```
int num = 31;
int val1 = (int) ((float) (num / 31 / 1 * 2 / 9) + (int) 1.0);
```


Primitive Types

- Type casting
  - Java casts types only when they don't match
  - Sample question:

```
int num = 31;
int val1 = (int) ((float) (num / 31 / 1 * 2 / 9) + (int) 1.0);
```

  - Answer: val1 is 1
    - num / 31 / 1 * 2 / 9 will keep int type, and the value is 0
    - They are in a pair of parentheses, and include only int variables
    - It is converted to float, but still with value 0
    - The remaining part is easy

Strings

- Recall Question 9 and Question 10 in the midterm

```
String str = "How are you?";
System.out.println(str.length() + ",” +
    str.equalsIgnoreCase("HOW ARE YOU") + "," +
    str.indexOf("ou“) + "," +
    str.lastIndexOf('a') + "," +
    str.charAt(6) + ",” +
    str.substring(1, 6));
```


Strings

- Recall Question 9 and Question 10 in the midterm

```
String str = "How are you?";
System.out.println(str.length() + ",” +
    str.equalsIgnoreCase("HOW ARE YOU") + "," +
    str.indexOf("ou“) + "," +
    str.lastIndexOf('a') + "," +
    str.charAt(6) + ",” +
    str.substring(1, 6));
```

  – The output: 12, false, 9, 4, e, ow ar

Strings

- *str.length()*
  - int type, the value is 12, not 11
- *str.equalsIgnoreCase("HOW ARE YOU")*
  - **boolean** type. The value can only be **true** or **false**
  - Think about str.equals(anotherString)
  - The answer is false, because the last '?' is missing.
  - str.equalsIgnoreCase("HOW ARE YOU?") will be true

Strings

- *str.indexOf("ou")*
  - int type, the value is 9
  - The value is not 9 and 10
    - An integer can not have two values
  - indexOf() can search for a single character, or a string
  - The first position where “ou” appears is 9

H	o	w		a	r	e		y	o	u	?
0	1	2	3	4	5	6	7	8	9	10	11

Strings

- *str.lastIndexOf(" ")*
  - int type, the value is 7
- *str.charAt(6)*
  - char type, the value is ‘e’
- *str.substring(1,6)*
  - String type, the value is “ow ar”

H	o	w		a	r	e		y	o	u	?
0	1	2	3	4	5	6	7	8	9	10	11

Strings

- Sample question:

```
String str2 = "Bananas are for monkeys“;
String val4 = str2.substring(str2.indexOf("n"), 6);
```


Strings

- Sample question:

```
String str2 = "Bananas are for monkeys“;
String val4 = str2.substring(str2.indexOf("n"), 6);
```

  - Answer: val4 is “nana”
    - indexOf(“n”) returns 2, which represents the first ‘n’
    - substring(2,6) returns 4 letters after the first ‘n’
    - It is easy to get “nana” in this question

Strings

- Sample question:

```
String str2 = "Bananas are for monkeys“;
String val2 = str2.substring(0, 1) + str2.substring(8, 12)
    + str2.substring(str2.indexOf("monkeys"));
```


Strings

- Sample question:

```
String str2 = "Bananas are for monkeys“;
String val2 = str2.substring(0, 1) + str2.substring(8, 12)
    + str2.substring(str2.indexOf("monkeys"));
```

  - Answer: val2 is “Bare monkeys”
    - Nothing complicated. Just remember that “+” means “to connect Strings”

Branch Statements – If and Else

- You can use only one if statement
  - ***if (boolean expression)***
    { statements; }
    other statements;
    - *Other statements* will always be executed
- You can also use an if-else statement
  - ***if (boolean expression)***
    { statements 1; }
    else { statement 2; }
    - If the *expression* is true, run *statement 1*, otherwise run *statement 2*

Boolean Expressions

- A combination of values and variables by comparison operators. Its value can only be ***true*** or ***false***

FIGURE 3.7 The Effect of the Boolean Operators && (*and*), || (*or*), and ! (*not*) on Boolean Values

Value of *A*	Value of *B*	Value of *A* && *B*	Value of *A* || *B*	Value of ! (*A*)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Boolean Expressions

- Sample question:

```
int num = 31;
boolean val3 = ((30 / num != 0) == (num % 15 >= 9));
```


Boolean Expressions

- Sample question:

```
int num = 31;
boolean val3 = ((30 / num != 0) == (num % 15 >= 9));
```

  - Answer: val3 is true
    - 30 / num is 0, 0 != 0 is false
    - num % 15 is 1 because 31=15*2+1. 1 >= 9 is false
    - false == false is true

Loop Statements

- While, do-while, for
  - You must expect that all loop-related questions now include arrays
  - There won't be complicated manipulations. However, you must be familiar with the execution orders of all parts in a loop

Loop Statements

- Sample question:
  - Write the output for:

```
int x = 7;
boolean found = false;

do {
    System.out.print(x + " ");
    if (x <= 2)
        found = true;
    else
        x = x - 5;
} while (x > 0 && !found);
```


Loop Statements

- Sample question:
  - Write the output for:

```
int x = 7;
boolean found = false;

do {
    System.out.print(x + " ");
    if (x <= 2)
        found = true;
    else
        x = x - 5;
} while (x > 0 && !found);
```

  - Answer: 7,2
    - In the first iteration, no condition is tested. 7 is the output, and x is set to 2
    - x > 0 and found is false, the second iteration starts, output 2 and set found as true
    - x > 0 but found is true. No more iteration will be executed

Loop Statements

- Sample question:
  - Write some code that will declare, initialize, and fill in an array of type int. After your code executes, the array should look as follows

0	2	4	6	8	10	12	14	16	18

Loop Statements

- Sample question:
  - Write some code that will declare, initialize, and fill in an array of type int. After your code executes, the array should look as follows

0	2	4	6	8	10	12	14	16	18

  - A "cheating" answer

```
int[] a = { 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 };
```


Loop Statements

- Sample question:
  - Write some code that will declare, initialize, and fill in an array of type int. After your code executes, the array should look as follows

0	2	4	6	8	10	12	14	16	18

  - Expected answer:

```
int[] b = new int[10];
for (int i = 0; i < 10; i++) {
    b[i] = 2 * i;
}
```


Arrays

- Sample question:
  - Given an array whose elements are in range [1,10]. Write a method to output how many each number appears in the array
    - Example: if the array is a = {3, 5, 3, 6, 8, 1, 1, 3};
    - count(a) should output:
      - 1 appears 2 times in the array
        3 appears 3 times in the array
        5 appears 1 times in the array
        6 appears 1 times in the array
        8 appears 1 times in the array

Arrays

- One possible answer:
  - Enumerate all possible values using nested loop

```
public static void count(int[] a) {
    for (int i = 1; i <= 10; i++) {
        int count = 0;
        for (int j = 0; j < a.length; j++) {
            if (a[j] == i) {
                count++;
            }
        }
        if (count > 0) {
            System.out.println(i + " appears " + count
                + " times in the array.");
        }
    }
}
```


Arrays

- Another possible answer:
  - Count all numbers with respect to an array count[i]

```
public static void count(int[] a) {
    int[] count = new int[11];
    for (int i = 1; i < 11; i++) {
        count[i] = 0;
    }
    for (int j = 0; j < a.length; j++) {
        count[a[j]]++;
    }
    for (int i = 1; i < 11; i++) {
        if (count[i] > 0) {
            System.out.println(i + " appears " + count[i]
                + " times in the array.");
        }
    }
}
```


Methods

- Sample question (parameters and return type):
  - Write a method header for methods that do each of the following things. Their headers start with the keywords public and static. Do not write the body of the method.
    - A method named printX() that just displays the String “X” to the output window.
    - A method named doubleValue() that takes in an argument of type int and returns twice the argument’s value.
    - A method named piCount() that takes in an array of doubles and returns the number of elements that are greater than Pi.
    - A method named largerThan() that takes in one int and one double and returns true if the int is larger than the double, and false otherwise.

Methods

- Answer:
  - public static void printX()
  - public static int doubleValue(int n)
  - public static int piCount(int[] a)
  - public static boolean largerThan(int i, double d)

Methods

- Sample question (local variables and return values):
  - Show the output produced by the following code

```
public class MyClass {

    public static void changeX() {
        int x = 20;
        System.out.println(x);
    }

    public static void incrementX(int x) {
        x++;
        System.out.println(x);
    }
```

```
public static int returnX(int x) {
    x = 0;
    System.out.println(x);
    return x;
}

public static void main(String[] args) {
    int x = 10;
    changeX();
    System.out.println(x);
    incrementX(x);
    System.out.println(x);
    x = returnX(x);
    System.out.println(x);
    }
}
```


Methods

- Sample question (local variables and return values):
  - Show the output produced by the following code

```java
public class MyClass {

    public static void changeX() {
        int x = 20;
        System.out.println(x);
    }

    public static void incrementX(int x) {
        x++;
        System.out.println(x);
    }
```

```java
public static int returnX(int x) {
    x = 0;
    System.out.println(x);
    return x;
}

public static void main(String[] args) {
    int x = 10;
    changeX(); // 20
    System.out.println(x); // 10
    incrementX(x); // 11
    System.out.println(x); // 10
    x = returnX(x); // 0
    System.out.println(x); // 0
    }
}
```


Next Lecture on Thursday

- Classes
- Inheritance
- Program 4

Classes

- Classes vs. objects
- Instance variables vs. static variables
- Methods with/without return values
- public/protected/private
- Class type variables (reference type)
- Constructors
- Method parameters – overloading
- Static variables and methods

Classes vs. Objects

- **Classes:**
  - What we can create
  - Specify the data to save
- **Objects**:
  - What have been created
  - Save actual data

Defining a class

```
public class Student
{
    public String name;
    public int classYear;
    public double GPA;
    public String major;
    // ...

    public String getMajor()
    {
        return major;
    }

    public void increaseYear()
    {
        classYear++;
    }
}
```

Class name

Instance Variables

Methods

Methods

```
public String getMajor()
{
    return major;
}

public void increaseYear()
{
    classYear++;
}
```

returns a String

return type

returns nothing

Methods with Parameters

```
public class Student
{
    public String name;
    public int classYear;
    // …
    public void setName(String studentName)
    {
        name = studentName;
    }
    public void setClassYear(int year)
    {
        classYear = year;
    }
}
```


Local Variable Rule

- Usually, a variable is only accessible in its surrounding brackets

```
public class Variable {
    String a = "a";

    public void f() {
        String b = "b";
        if (a.equals("b")) {
            String c = "c";
        }
    }
}
```


public/private Modifier

- public: there is no restriction on how you can use the method or instance variable
- private: can not directly use the method or instance variable's name outside the class
- protected: can not directly use the method or instance variable's name outside the class, except in the class's subclasses

public/private Modifier

```
public class Student
{
    public int classYear;
    private String major;
}
public class StudentTest{
   public static void main(String[] args){
        Student jack = new Student();
        jack.classYear = 1;
        jack.major = "Computer Science"; // ERROR!!!
   }
}
```

OK, *classYear* is *public*

Error!!! *major* is *private*

Well Encapsulation

- Imagine a wall between (other) programmers and (your) implementation
  - It's called interface

Implementation:

Private instance variables
Private constants
Private Methods
Bodies of all methods
Method definitions

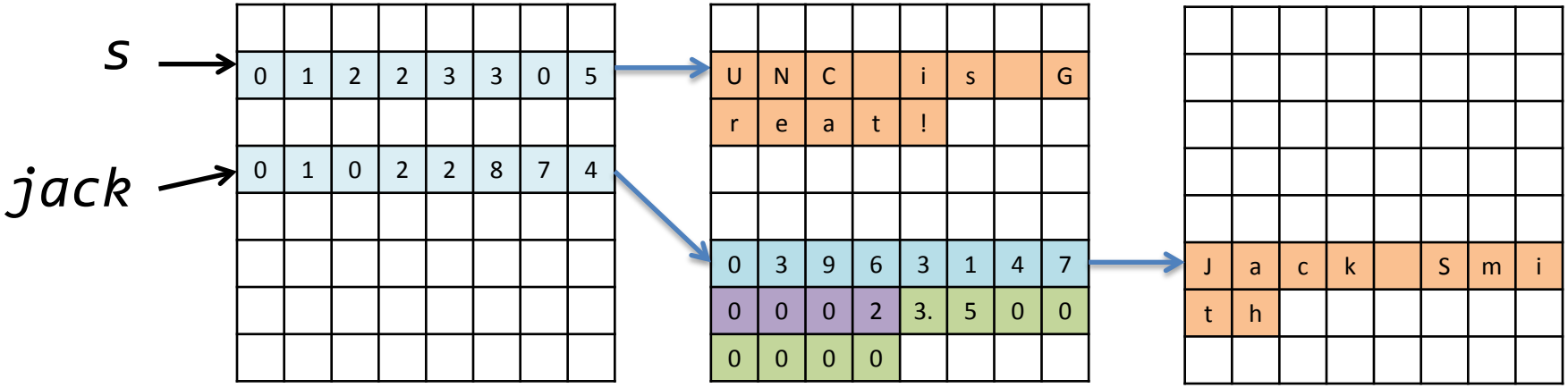
Interface:

Comments
Headings of public methods
Public defined constants

Programmer

Variables of a Class Type

- What goes in these variables?
  - In a class type variable, the address pointing to the actual object is saved (not the object itself)



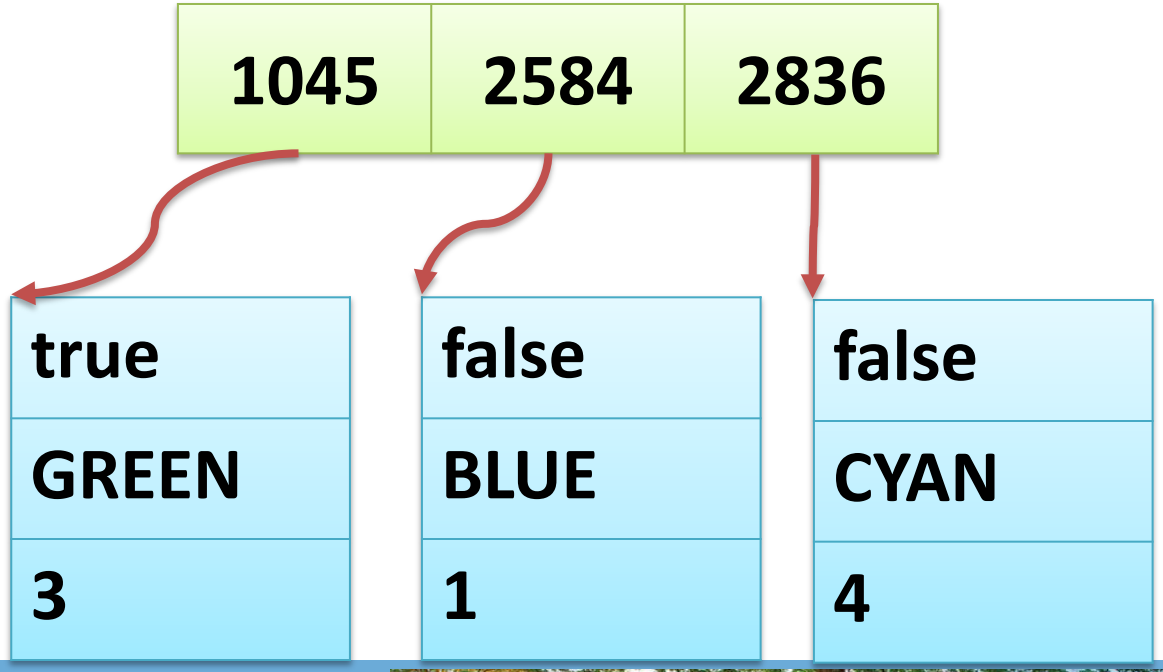
Array is Also a Class Type

var name	score[0]	score[1]	score[2]	score[3]	score[4]
data	62	51	88	70	74
m address	25131	25132	25133	25134	25135

- Index numbers start with **0**. They do NOT start with 1 or any other number.
- The array name represents a memory address, and the ith element can be accessed by the address plus i

Arrays of Class Types

```
Smiley[] smilies = new Smiley[3];
for (int i = 0; i < smilies.length; i++) {
    smilies[i] = new Smiley();
}
```



Key Message of Class Types

- A primitive type can **never** be changed by being passed to a method as a parameter
  - It is impossible to change x like this:
    - int x = 10;
      incrementX(x);
- A class type's contents can be changed by passing to a method
    - int[] a = new int[5];
      swap(i,j,a);

Constructor

- A special method with the same name as the class, and no return type
- Called only when an object is created
- It can take parameters to initialize instance variables
- You can define multiple constructors with different parameter lists

Example: Pet class

```
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet()
    {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public static void main(String[] args)
    {
        Pet p = new Pet();
    }
}
```

Default constructor

Call constructor

Constructors with Parameters

- If you define at least one constructor, a default constructor will **not** be created for you
- Now you **must** create a Pet object like this:
  - `Pet odie = new Pet("Odie", 3, 8.5);`
  - `Pet odie = new Pet(); // WRONG! No default constructors!`

```
public class Pet {
    private String name;
    private int age;
    private double weight;
    public Pet(String initName, int initAge, double initWeight)
    {
        name = initName; age = initAge;   weight = initWeight;
    }
}
```


Method Overloading

- Overloading means several methods share the same name but have different parameters
- Java calls the methods according to the parameter numbers and types
  - The name, parameter number and parameter type form the method signature
- Make sure that they do the same thing. Otherwise the user will be confused

Methods Overloading

- We've seen that a class can have multiple constructors. Notice that they have the same name

```
public class Pet {
    public Pet() {…}
    public Pet(String initName, int initAge, double initWeight)
    {…}
    public Pet(String initName) {…}
    public static void main(String[] args) {
        Pet p = new Pet(); // First constructor will be called
        Pet q = new Pet("Garfield", 3, 10); // Second constructor
        Pet w = new Pet("Odie"); // Third constructor
        Pet u = new Pet("Nermal", 2); // Wrong – no matching method
    }
```


Static Variables/Methods

- Static variables and methods belong to a class instead of an object
- Every object has its own instance variables; all objects in the same type share the same static variables
- Pay attention to: what can be accessed in different methods

Example: Static Variables and Methods

```java
public class Pet {
	private String name;
	private static int totalNumber = 0;
	// totalNumber is initialized when the first object is created

	public Pet(String initName) {
		this.name = initName;
		// Recommended: use "this" to call instance variables
		totalNumber++; // totalNumber can be accessed in an instance method
		System.out.println("Total pet number is " + Pet.getTotalNumber());
		// Recommended: use class name to call static variables
	}

	public static int getTotalNumber() {
		return totalNumber;
		// You can not access "name" or "this" in a static method
	}

	public static void main(String[] args) {
		Pet a = new Pet("Odie");
		Pet b = new Pet("Garfield");
		Pet c = new Pet("Nermal");
		// Three objects are created, so totalNumber is increased for three times
		System.out.println("Total pet number is " + a.getTotalNumber());
		System.out.println("Total pet number is " + b.getTotalNumber());
		// You can invoke a static method from an object. However they perform the same.
		// You are recommended to call it as Pet.getTotalNumber();
	}
}
```


Inheritance

- What is inheritance
  - Subclasses inherit all public and protected variables and methods from superclass
- What is overriding
  - If a subclass defines a method of the same **signature** as the super class, this is ***overriding***
- What is polymorphism
  - A subclass object can be assigned to a superclass variable
  - It can perform its own action from overridden methods

Polymorphism and Overriding

```java
public class Animal {
    private String animalName;
    public void speak() {
    // default method -- can be empty
    }

    public static void main(String[] args)
    {
        Animal a[] = new Animal[3];
        a[0] = new Cat();
        a[1] = new Dog();
        a[2] = new Duck();
        for (int i = 0; i < 3; i++) {
            a[i].speak();
        }
    }
}
```

```java
public class Cat extends Animal {
    public void speak() {
        System.out.println("MEW");
    }
}

public class Dog extends Animal {
    public void speak() {
        System.out.println("WOOF");
    }
}

public class Duck extends Animal {
    public void speak() {
        System.out.println("QUACK");
    }
}
```

Output: MEW, WOOF, QUACK

The *is-a* Relationship

```java
public class Animal {
    public void eat() {
        System.out.println("Get
            anything to eat");
    }
}

public class Mammal extends Animal {
}

public class Bear extends Mammal {
    public void eat() {
        System.out.println("Find a
            fish to eat");
    }
    public void hibernate() {
        System.out.println("Zzzzzz");
    }
}
```

```java
public static void main(String[]
args) {
    Animal a = new Mammal();
    // YES! A Mammal is an Animal
    Animal b = new Bear();
    // YES! A Bear is an Animal
    Mammal c = new Bear();
    // YES! A Bear is a Mammal
    // Bear d = new Mammal(); NO! A
    // Mammal may not be a Bear!
    a.eat(); // OK. Mammal doesn't
    // override eat(). Eat anything.
    b.eat(); // OK. Bear overrides
    // eat(). Eat fish.
    // c.hibernate(); WRONG! Mammal
    // doesn't have this method!
}
```


Sample Question

- Write two classes to inherit a given class Person
  - Person represents a person working in the university
    - It has 3 protected instance variables: hourlyRate, hoursPerWeek and insuranceCost
    - Also, one static variable: WEEKSPERSEMESTER
  - Student represents a student who works in part-time
  - Employee represents a permanent employee
  - You must override getIncome() and getOutcome() methods to generate correct output

Sample Question

- Write two classes to inherit a given class Person
  - You must override getIncome() and getOutcome() methods to generate correct output
    - A student's income is: hourly rate * hours per week * week per semester
    - A student's outcome is: tuition cost + insurance cost
    - An employee's income is: base salary + hourly rate * hours per week * week per semester
    - An employee's outcome is: insurance cost
  - Write getTotalBalance() to calculate income – outcome
  - The expected output is given

Solution to Sample Question

- The getTotalBalance() method in Person

```
public double getTotalBalance() {
    return this.getIncome() - this.getOutcome();
    // getIncome() and getOutcome() are implemented
        in subclasses -- but it is fine
}
```


Solution to Sample Question

- The Student class

```
class Student extends Person {
      private double tuitionCost;

      public Student(double tuition, double rate, int hours, double insurance) {
            super(rate, hours, insurance);
            // You must use super() to call superclass's constructor
            this.tuitionCost = tuition;
            // tuitionCost must be initialized
      }

      // getIncome() and getOutcome() must be implemented
      public double getIncome() {
            return this.hourlyRate * this.hoursPerWeek * Person.WEEKSPERSEMESTER;
            // hourlyRate and hoursPerWeek are inherited. WEEKSPERSEMESTER can be called directly
      }

      public double getOutcome() {
            return this.tuitionCost + this.insuranceCost;
            // tuitionCost is newly defined.
      }
}
```


Solution to Sample Question

- The Employee class

```
class Employee extends Person {
      private double baseSalary;

      public Employee(double base, double rate, int hours, double insurance) {
            super(rate, hours, insurance);
            this.baseSalary = base;
      }

      public double getIncome() {
            return this.baseSalary + this.hourlyRate * this.hoursPerWeek
                  * Person.WEEKSPERSEMESTER;
      }

      public double getOutcome() {
            return this.insuranceCost;
      }
}
```


Closing Note

- It is my great pleasure to have all of you in the class
  - I hope that you enjoyed this course
- I will appreciate if you take the online evaluation
- My doctorate dissertation defense is on 9:00am tomorrow, at FB 141
  - I will start working as a senior software engineer at MathWorks in this summer
- **Thank you for taking this course!**