

CSCI 136 Written Exam #0
Fundamentals of Computer Science II
Spring 2013

Name: _____

This exam consists of 5 problems on the following 7 pages.

You may use your single-side hand-written 8 ½ x 11 note sheet during the exam. You may use a simple handheld calculator. No computers, mobile devices, cell phones, or other communication devices of any kind are permitted.

If you have a question, raise your hand and I will stop by. Since partial credit is possible, **please write legibly and show your work.**

Problem	Points	Score
1	8	
2	15	
3	12	
4	12	
5	12	
Total	59	

1. **Loops, Input, Exceptions** (8 points). Consider the following program:

```
public class Prob1
{
    public static void main(String[] args)
    {
        int i = 0;
        double d = 0.0;
        for (String s : args)
        {
            try
            {
                d += Double.parseDouble(s);
                i++;
            }
            catch (NumberFormatException e)
            {
            }
        }
        System.out.printf("%d / %d : %.2f\n", i, args.length, d);
    }
}
```

Below are four example executions of the program. Give the output produced by the program. If the given input would cause a runtime error, write "runtime error". If it would cause a compile error, write "compiler error".

Command line	Output
% java Prob1 1.0 2.0 0.5	
% java Prob1 -1 -2 -0.5	
% java Prob1 1.0 two 0.5	
% java Prob1	

2. **Inheritance, Collections** (15 points). Consider the class hierarchy given on the last page of the exam.

a) Which data types could you successfully create with the new operator? Circle **all** that apply.

- I. Person
- II. Student
- III. Employee
- IV. Faculty
- V. Staff

b) Assume someone has declared an array of type Employee. Concrete objects of what types could be stored in this array? Circle **all** that apply.

- I. Person
- II. Student
- III. Employee
- IV. Faculty
- V. Staff

c) The toString() method in the class Faculty is an example of what? Circle the **single best** answer.

- I. Overloaded method
- II. Overridden method
- III. Polymorphism
- IV. Auto-boxing
- V. Ducking an exception

d) The constructors in the class Staff are an example of what? Circle the **single best** answer.

- I. Polymorphic methods
- II. Abstract methods
- III. Overloaded methods
- IV. Overridden methods
- V. Auto-unboxed methods

e) Consider the following code snippet:

```
ArrayList<Faculty> list = new ArrayList<Faculty>();  
// ... (code that adds a bunch of faculty members)  
Collections.sort(list); // sort by number of advisees
```

You get the following compile error:

```
Bound mismatch: The generic method sort(List<T>) of type Collections is not  
applicable for the arguments (ArrayList<Faculty>). The inferred type Faculty is  
not a valid substitute for the bounded parameter <T extends Comparable<? super T>>
```

What do you need to do to the Faculty class to make the sort work? Circle **all** that apply.

- I. Add "implements Comparable<Faculty>" to the class declaration.
- II. Add "implements Comparable<ArrayList<Long>>" to the class declaration.
- III. Add "implements Comparable" to the declaration of the Faculty class.
- IV. Add "implements Comparable<Integer>" to the class declaration.
- V. Add a method "public int compareTo(Person other)"
- VI. Add a method "public void sort(ArrayList<Faculty>)"
- VII. Add a method "public int compareTo(Faculty other)"

f) Various instance variables are declared in the class hierarchy. Put a checkmark in the boxes corresponding to the instance variables (left column) that are directly accessible in instance methods of each class (top row). That is, instance variables that could be used without going through another method.

	Person	Student	Employee	Faculty	Staff
name					
id					
gpa					
credits					
salary					
sickDays					
advisees					
tenured					
safety					

g) Each Faculty object keeps track of the list of student ID's that faculty member is responsible for. Advisees can be added and removed one-at-a-time by ID. Add code to the following three methods to correctly implement this functionality:

```
public Faculty(String n, int sal, boolean ten)
{
    super(
        );

    this.tenured = ten;

    advisees =

}

public void addAdvisee(long id)
{

}

public boolean removeAdvisee(long id)
{

}

}
```

3. Loops, Conditionals, String (12 points). A school converts letter grades to a value as follows:

A	A-	B+	B	B-	C+	C	C-	D	F
4.0	3.7	3.3	3.0	2.7	2.3	2.0	1.7	1.0	0.0

A student's GPA (grade point average) is computed by using these floating-point values weighted by the number of credits of the class in which each grade was earned. So for example, if a student has completed two classes, getting an A in a 4-credit class and a B- in a 3-credit class, his GPA is:

$$(4 * 4.0 + 3 * 2.7) / 7 = 3.442857$$

a) Complete the following helper method that converts a String containing a letter grade to its floating-point value. Your method should work regardless of whether the grade is in upper- or lower-case. Recall that the String data type has methods toUpperCase() and toLowerCase(). If the passed in grade is not a grade in the above table, throw a RuntimeException indicating "Invalid grade".

```
private double convertGrade(String grade)
{
    String [] grades = {"A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D", "F"};
    double [] val     = {4.0, 3.7, 3.3, 3.0, 2.7, 2.3, 2.0, 1.7, 1.0, 0.0};

}
```

b) Using your convertGrade() method, implement the addCourse() method in the Student class from the diagram on the last page of the exam. Your method should result in updating the object's instance variables so as to reflect the student's new GPA and total credits.

```
public void addCourse(String grade, int credits)
{

}
```

4. Exceptions, Collections (12 points).

a) Java has both checked and unchecked exceptions. Briefly describe the difference. Provide an example of a typical exception of each type.

b) Describe a problem where a **list** abstract data type (ADT) would be most appropriate. Why is a **list** the best choice for this problem?

c) Describe a problem where a **map** abstract data type (ADT) would be most appropriate. Why is a **map** the best choice for this problem?

d) Describe a problem where a **set** data type (ADT) would be most appropriate. Why is a **set** the best choice for this problem?

5. Methods (12 points). Match the method description on the left with the **best** method on the right. Assume candidate methods appear inside a class named `Book`. The `Book` class knows things like the author, title, and all the text of a book (the exact details are not important). Not every letter will be used. Each letter will be used **at most once**.

_____ A method that can determine if a `Book` contains exactly the same text as another `Book`.

_____ You have an object `c` of type `Book`. This method is called implicitly by the following line:
`System.out.println(c);`

_____ A method that can calculate how many times each word in a set of words appears in a `Book`. For example, the method might calculate that “whale” occurs 10 times and “ocean” appears 17 times.

_____ A copy constructor that creates a `Book` based on another `Book`.

_____ A method that can return the count of the number of words that overlap in two `Book` objects.

_____ A method that can return the set of possible misspelled words in a `Book` given a passed in dictionary of known good words.

A. **public** `String toString()`

B. **public static** `String toString()`

C. **public void** `toString(String s)`

D. **public boolean** `foo(String s)`

E. **public boolean** `foo(Book a)`

F. **public boolean** `foo()`

G. **public int []** `foo(String [] s)`

H. **public int** `foo(ArrayList<String> s)`

I. **public int** `foo(HashSet<String> s)`

J. **public int** `foo(Book a)`

K. **public** `HashSet<String> foo(HashSet<String> s)`

L. **public** `String [] foo()`

M. **public void** `foo(String s)`

N. **public** `Book()`

O. **public** `Book(Book a)`

P. **public** `Book(other)`

Q. **private** `Book(Book a)`

This page intentionally left blank.


```
public abstract class Person {
    protected String name; // full name
    public Person(String name) {...}
    public abstract String toString();
}
```

```
public class Student extends Person {
    protected long id; // unique id
    protected double gpa; // grade point average
    protected int totalCredits; // credits earned
    public Student(String n, double gpa, long id) {...}
    public String toString() {...}
    public void addCourse(String grade, int credits) {...}
}
```

```
public abstract class Employee extends Person {
    protected int salary; // salary in dollars
    private int sickDays; // accumulated sick days
    public Employee(String name, int salary) {...}
    public int getSickDays() {...}
    public String toString() {...}
}
```

```
public class Faculty extends Employee {
    protected ArrayList<Long> advisees; // advisee IDs
    protected boolean tenured; // has tenure?
    public Faculty(String n, int salary, boolean ten) {...}
    public void addAdvisee(long id) {...}
    public boolean removeAdvisee(long id) {...}
    public String toString() {...}
}
```

```
public class Staff extends Employee {
    protected boolean safety; // safety officer?
    public Staff(String n, int sal) {...}
    public Staff(String n, int sal, boolean safe) {...}
}
```

