University of Maryland College Park
Department of Computer Science
CMSC132 Fall 2019
Exam #1 Key

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g. 123456789):

Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 200 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- For multiple choice questions you can assume only one answer is expected, unless stated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

Grader Use Only

#1	Problem #1 (Miscellaneous)	36	
#2	Problem #2 (Class Implementation)	164	
Total	Total	200	

Problem #1 (Miscellaneous)

1. (3 pts) Which of the following describes data abstraction? Circle all that apply.

    a. Specifies actions that should be performed, hiding any algorithms.
    b. Specifies actions that should be performed and the objects for the problem, hiding any algorithms or data representation.
    c. None of the above.

    Answer: c.

2. (3 pts) Encapsulation is a design technique for hiding:

    a. Implementation details.
    b. Documentation for a class.
    c. The speed associated with software.
    d. None of the above.

    Answer: a.

3. (3 pts) In a constructor for a subclass, if you don't call a superclass constructor using super:

    a. No superclass constructor will be called.
    b. The superclass default constructor will be called.
    c. The constructor of the superclass that has at least one parameter will be called.
    d. None of the above.

    Answer: b.

4. (3 pts) In static (early) binding, the decision of which method to call takes place at:

    a. Run time.
    b. Compile time.
    c. Either run time or compile time.
    d. None of the above.

    Answer: b.

5. (3 pts) What will take place when the following code fragment is executed?

    Object a = null;
    boolean val = a instanceof Object;

    a. An exception will be generated.
    b. false will be assigned to val.
    c. true will be assigned to val.
    d. None of the above.

    Answer: b.

6. (3 pts) A method called **task** throws a checked exception called **IllegalBase**. How would you update the method so we do not have to add a try/catch block?

```
public void task(){
    // code that throws checked exception
}
```

Answer: public void task() **throws IllegalBase {**

7. (8 pts) Given the classes below, indicate whether the assignments are valid or invalid. Notice that we are using two packages.

```
package toyPackage;
public class Toy {
   protected int size;
   static int max;
   public static final int temp = 10;
}


package experiment;
import toyPackage.*;
public class Driver {
   public static void main(String[] args) {
      Toy p = new Toy();

      p.size = 10;   /* Invalid (2 pts) */

      p.max = 20;    /* Invalid (2 pts) */

      Toy.max = 30;  /* Invalid (2 pts) */

      Toy.temp = 40; /* Invalid (2 pts) */
   }
}
```

8. (10 pts) A class called **Printer** is defined as follows:

```
public class Printer {
   private String id;
   private int pages;
   private char type;

   public Printer(String id, int pages) {
      this.id = id;
      this.pages = pages;
      this.type = 'R';
   }

   public int increasePages(int delta, char type) {
      this.type = type;
      return pages + delta;
   }

   public final void printPage(int n) {
      pages -= n;
   }
}
```

Another class called **LaserPrinter** extends **Printer**. Indicate what will happen if one of the methods below were to be added to the **LaserPrinter** class. Circle RIDE to indicate the method will override a method in the Printer class, LOAD to indicate it will overload a method, and ERROR if it will generate a compilation error. Notice that you should consider each of them individually (assume you only are adding a., b., etc.) when answering each item.

```
a. public void increasePages(int delta, char type) { }           ERROR (2 pts)
b. public int increasePages(char type, int delta) { return 1; }  LOAD  (2 pts)
c. public int increasePages(int delta, char type) { return 1; }  RIDE  (2 pts)
d. public void printPage(int n) { }                              ERROR (2 pts)
e. public void printPage() { }                                   LOAD  (2 pts)
```

Problem #2 (Class Implementation)

For this problem you will complete the implementation of the **Book** and **ElectronicBook** classes (whose partial definitions are provided below). A **Book** object has a title, cost and rating (high rating values imply a better book). An **ElectronicBook** object is book that has a field (megaBytes) indicating the memory space required by the book. Methods for both classes are public, non-static methods, unless specified otherwise. Y**ou may not add any instance nor static variables and you may not add any auxiliary methods.**

Below you will see a sample driver and expected output that illustrates the functionality of the classes you need to implement. The driver relies on methods and classes you don't need to implement (e.g., toString()). Feel free to ignore the driver if you know what to implement.

```
public class Book implements Comparable<Book> {
   private String title;
   private int cost, rating;
   private static int instances = 0;

   /* INCOMPLETE CLASS */
}
```

```
public class ElectronicBook extends Book {
   public int megaBytes;

   /* INCOMPLETE CLASS */
}
```

Sample Driver / Output

```
Book b1 = new Book("Java1", 20, 4), b2 = new Book("More Java", 20, 5);
System.out.println(b1 + "\n" + b2);
System.out.println(b2.compareTo(b1) < 0 ? "***First" : "***Second");

ElectronicBook eb1 = new ElectronicBook("SuperJava", 34, 3, 50);
ElectronicBook eb2 = new ElectronicBook("C Language", 17, 2, 100);
System.out.println(eb1 + "\n" + eb2);

ArrayList<Book> list = new ArrayList<Book>();
list.add(b1);
list.add(eb1);
list.add(b2);
list.add(eb2);
System.out.println("Average: " + Utilities.getAvgMegaBytes(list));
```

```
Title: Java1, Cost: 20, Rating: 4
Title: More Java, Cost: 20, Rating: 5
***First
Title: SuperJava, Cost: 34, Rating: 3,
Mega: 50
Title: C Language, Cost: 17, Rating: 2,
Mega: 100
Average: 75
```

1. **Book Class Methods**

   a. **Constructor -** It has as parameters the title, cost and rating. It will initialize the corresponding instance variables. In addition it will update the **instances** variable so we can keep track of how many **Book** objects have been created. The maximum number of **Book** objects we can have is 1000. If the constructor is called when the number of instances is 1000, the exception **TooManyException** will be thrown and no further initialization will take place. The parameter for this exemption is the string "too many".

   Answer:

```
public Book(String title, int cost, int rating) {
       if (instances > 1000) {
              throw new TooManyException("too many");
       } else {
              this.title = title;
              this.cost = cost;
              this.rating = rating;
              instances++;
       }
}
```

   b. **Default Constructor** – Initializes a **Book** object with a title value that corresponds to "NOTITLE" and a value of 0 for both the **cost** and **ratings**. You must call the previous constructor in order to implement this one, otherwise you will not get credit.

   Answer:

```
public Book() {
       this("NOTITLE", 0, 0);
}
```

   c. **compareTo –** Books will be compared based on their **cost**, so books with lower cost will appear first if they were to be sorted. If two books have the same cost, a book with a higher rating will precede a book with a lower one.

   Answer:

```
public int compareTo(Book book) {
       int comp = cost - book.cost;
       if (comp != 0) {
              return comp;
       } else {
              return book.rating - rating;
       }
}
```

2. **ElectronicBook Class Methods**

   a. **Constructor -** It has as parameters the title, cost, rating and megabytes associated with the book. The method will initialize the object based on the provided parameters.

   Answer:

```
public ElectronicBook(String title, int cost, int rating, int megaBytes) {
       super(title, cost, rating);
       this.megaBytes = megaBytes;
}
```

b. **getMegaBytes –** Returns the **megaBytes** value.

Answer:

```
public int getMegaBytes() {
       return megaBytes;
}
```

3. **getAvgMegaBytes static method –** Implement a static method named **getAvgMegaBytes** (see prototype below) that is part of an **Utilities** class. The method will return the average number of megabytes for electronic books that are in the ArrayList parameter. Notice the method returns an integer. You can assume the parameter will not be null.

Answer:

```
public static int getAvgMegaBytes(ArrayList<Book> books) {
       int sum = 0, count = 0;

       for (Book book : books) {
              if (book instanceof ElectronicBook) {
                     sum += ((ElectronicBook) book).getMegaBytes();
                     count++;
              }
       }

       return sum / count;
}
```