University of Maryland College Park
Department of Computer Science
CMSC132 Fall 2021
Exam #1

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 100 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)	40	
#2	Problem #2 (Class Implementation)	60	
Total	Total	100	

Problem #1 (Miscellaneous)

1. (3 pts) Calling `Collections.sort()` in your code would be an example of ______

   a. using data abstraction
   b. using procedural abstraction
   c. invoking an abstract method of an interface that has been implemented
   d. None of the above

2. (3 pts) Assume a `csStudent` class implements a `Person` Interface and extends a `Student` class. Which is not true?

   a. `csStudent` "is a" `Student`
   b. `csStudent` "is a" `Person`
   c. A `csStudent` object can be assigned to a variable of type `Person`
   d. `Student` "is a" `csStudent`

3. (3 pts) Which statement is false.

   a. You can have an abstract class without any abstract methods.
   b. If a class is missing a default constructor (i.e. one that takes not argument), it cannot be a base class.
   c. If a class is declared as final it can inherit from a non-final base class.
   d. An Enum type can have a private constructor.

4. (3 pts) In no more than 2 sentences, explain what is meant by static (early) binding.

5. (3 pts) Assume a `csStudent` class implements a `Person` Interface and extends a `Student` class . What will take place when the following code fragment is executed?

```
Object o = new Student();
boolean b = o instanceof csStudent;
```

   a. An exception will be generated.
   b. false will be assigned to b.
   c. true will be assigned to b
   d. It will not even compile.

6. (3 pts) What Java feature is most prominently being demonstrated here?

```
ArrayList <Integer> myList = new ArrayList <Integer>();
myList.add(5);
```

   a. autoboxing
   b. unboxing
   c. inheritance
   d. interfaces

7. (12 pts) Given the classes below, indicate whether the assignments are valid or invalid. Notice that we are using two packages.

```
package packA;

public class Base {
       int packV;
       protected int protV;
       final static int myVar = 10;  }

--------------------

package packA;

public class NotAChild {

public static void main(String[] args) {
       Base b = new Base();

       b.packV = 1;  /* Valid or Invalid (Circle your choice) */

       b.protV = 1;  /* Valid or Invalid (Circle your choice) */

       Base.myVar = 1;  /* Valid or Invalid (Circle your choice) */
}}

--------------------

package packB;

import packA.Base;

public class Child extends Base{

public static void main(String[] args) {
       Child c = new Child ();

        c.packV = 1;  /* Valid or Invalid (Circle your choice) */

       c.protV = 1;  /* Valid or Invalid (Circle your choice) */

       Base.myVar = 1;  /* Valid or Invalid (Circle your choice) */

}}
```

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

```
package packA;

public class Student {

  void someMethodA (int x)
  {
         //some code
  }

}
```

Another class called **csStudent** extends **Student**. Indicate what will happen if one of the methods below were to be added to the **csStudent** class. Circle RIDE to indicate the method will override a method in the Student 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 someMethodA (int x){}`	**RIDE / LOAD / ERROR**
b.	`void someMethodA (int x){}`	**RIDE / LOAD / ERROR**
c.	`private void someMethodA (int x){}`	**RIDE / LOAD / ERROR**
d.	`public int someMethodA (int x){return 0;}`	**RIDE / LOAD / ERROR**
e.	`public void someMethodA (String x){}`	**RIDE / LOAD / ERROR**

DirectoryId:

Problem #2 (Class Implementation)

Assume the following 2 classes that you should not change.

```
public abstract class Gadget {
        private int id;
        public Gadget(int id) {
                this.id = id;}
        @Override
        public String toString() {
                return "Gadget [id=" + id + "]";}
        public abstract String howToUse();
}
```

```
public class PortableTimeGadget extends TimeGadget {

public PortableTimeGadget(int id, int hour, int min) {
                super(id,  hour, min);
        }

}
```

For this problem you will complete the implementation of the `TimeGadget`, `TimeComparator`, and `Util` classes (whose partial definitions are provided below). A `TimeGadget` object has a `hour` and `min` field. The other 2 do not have fields. All 5 classes are in the same package. Y**ou may not add any instance nor static variables and you may not add any auxiliary methods to the classes.**

```
public class TimeGadget extends Gadget{

        private int hour;
        private int min;

        public int getHour() {
                return hour;
        }

        public int getMin() {
                return min;
        }
         /* INCOMPLETE CLASS */
}
```

```
import java.util.Comparator;
public class TimeComparator implements Comparator<TimeGadget> {

/* INCOMPLETE CLASS */
}
```

```
import java.util.ArrayList;

public class Util {

/* INCOMPLETE CLASS */
}
```

Below you will see a sample driver and expected output that illustrates the functionality of the classes you need to implement.

Sample Driver / Output

```
import java.util.ArrayList;
import java.util.Arrays;
public class SampleDriver {
        public static void main(String[] args) {
                TimeGadget t1 = new TimeGadget(35, 7, 17);
                TimeGadget t2 = new TimeGadget(36, 7, 18);
                TimeGadget t3 = new TimeGadget(37, 7, 17);
                TimeComparator timeComparator = new TimeComparator();

                int [] result = {timeComparator.compare(t1, t2),timeComparator.compare(t1, t3),
                timeComparator.compare(t2, t3)};

                System.out.println(Arrays.toString(result));

                Gadget t4 = new TimeGadget(38, 12, 3);
                Gadget t5 = new PortableTimeGadget(39, 2, 45);
                ArrayList <Gadget> g = new ArrayList <Gadget>();
                g.add(t1); g.add(t2); g.add(t3); g.add(t4); g.add(t5);
                Util.demo();
                System.out.println(Util.makeList(g));} }
```

```
[-1, 0, 1]
Bad Time
[Gadget [id=35] 7:17, Gadget [id=36] 7:18, Gadget [id=37] 7:17, Gadget [id=38] 12:03]
```

1. **TimeGadget Class Methods**

   a. **Constructor -** It has as parameters the id, hour, and min. It will call the base class constructor to set the id. If the hour is outside of the range of 1 to 12 (inclusive) or the min is outside of the range 0 to 59 (inclusive) it will throw the IllegalArgumentException with the message `Bad Time`, otherwise it will initialize the corresponding instance variables.

   b. **getTime** – The public non-static `getTime` method will return a String and take in no parameters. The return string will be the `hour` followed by a : followed by the `min`. If the min (not the hour) is less than 10, it should be padded with a zero. Therefore, 7:03 is valid but 7:3 is not when `hour` is 7 and `min` is 3.

DirectoryId:

c. **howToUse –** The public non-static `howToUse` method will return a String and take in no parameters. The return string will be simply be `Set time and look at it`. You must have the Override annotation before the method definition.

2. **TimeComparator Class Method**

a. **compare -** The public non-static `compare` method has parameters `TimeGadget t1`, `TimeGadget t2` and will return 1 if t1 is greater than t2, 0 if they are equal, and -1 if t1 is less than t2. As for ordering based on time, 1:00 is the smallest time and 12:59 is the largest.

3. **Util Class Methods**

a. **demo –** The purpose of this method is to demonstrate basic exception handling. `demo` is void and has no parameters. Simply create a `TimeGadget` object with 5 for the id, and 5 for the hour, and -17 for the min. When the exception is caught it will simply print out the message that the exception was created with in the `TimeGadget` constructor. You can obviously use a `System.out.println` call in your code, but if you are literally writing `System.out.println ("Bad Time")` in your code you are doing it wrong. Write the answer on the next page.

b. makeList – The method below will return an Arraylist of Strings where the String elements are the concatenation of calling the `toString`, a space, and calling `getTime` of **<u>only</u>** objects created using the `TimeGadget` constructor not `PortableTimeGadget` . Here is the challenge, you cannot use the `getClass` method, but can use any java operator you want. Please do not ask if such and such is an operator, we are testing you on that.

```
public static ArrayList<String> makeList(ArrayList<Gadget> gadget)
```

DirectoryId:

EXTRA PAGE IN CASE YOU NEED IT (SUBMIT WITH THE EXAM)

LAST PAGE