Name (last name ALL CAPS): **NetID**:

Grades for the final will be posted on the CMS some time tomorrow. Course grades will be posted next week. You can look at your final only when you return in the fall. HAVE A GOOD SUMMER!

Submit all regrade requests by 8PM TONIGHT. Use the CMS where possible. Regrades for prelims will not be considered.

You have 2.5 hours to complete the questions in this exam, which are numbered 0..7. Please glance through the whole exam before starting. The exam is worth 100 points.

Question 0 (2 points). Print your name and Cornell **net id** at the top of each page. Please make them legible.

The first few questions deal with an abstract class `Storage` and subclasses of it, for manipulating hard drives and their contents. When answering these questions, read carefully, look at the pertinent class invariants and specifications of methods, and deal with one issue at a time. Class `Storage` and class `MP3Song` appear below. They will be referred to later.

Question 0.	_____ (out of 02)
Question 1.	_____ (out of 15)
Question 2.	_____ (out of 15)
Question 3.	_____ (out of 13)
Question 4.	_____ (out of 13)
Question 5.	_____ (out of 14)
Question 6.	_____ (out of 14)
Question 7.	_____ (out of 14)
Total	_____ (out of 100)

```
/** An instance represents a storage device with a capacity (number of bytes it can hold) */
public abstract class Storage {
    private int cap;     // capacity of this storage device

    /** Constructor: an instance with capacity c */
    public Storage(int c) {
        cap= c;
    }

    /** = the capacity of this storage device */
    public int getCapacity() {
        return cap;
    }

    /** = amount of unused space */
    public abstract int remainingSpace();
}
```

```
/* An instance consists of an mp3 file and a title for it */
public class MP3Song {
    public Object mp3; // the song

    public String title; // title of song, ending in .mp3

    /** An mp3Song given in file f with title n.
        Throw an IllegalArgumentException if the title does not end in ".mp3" */
    public MP3Song(Object f, String n) {




    }
}
```

Question 1 (15 points) Exceptions, abstract classes, and subclasses.

(a) On the back of the previous page, write a class `OutOfSpaceException`, whose objects can be thrown. You determine which class it should extend.

(b) Abstract classes and subclasses. Below (use the back of the previous page if necessary), write a subclass `HardDrive` of class `Storage` (see page 1). *In addition to any components required from class `Storage`*, `HardDrive` should contain the following fields and methods; make them private or public according to standard practice. You do not have to put specs on the methods because they are given here:

- **`int`** `used` (a field): The amount of space used on the drive at the moment.
- `Vector contents` (a field): the objects that are on the drive. The **location** of an object on the hard drive is its index in this vector.
- A constructor that helps create an empty drive (nothing on it) of capacity c (a parameter).
- `add(Object ob, int s)`: A function to add object `ob` of size `s` to this hard drive and return its location. Throw an `OutOfSpaceException` if there is not enough space on the hard drive for ob.
- `format()`: A procedure to erase the contents of this hard drive.
- `get(int i)`: a function that returns the object on the hard drive that is at location i.

Question 2 (15 points). Subclasses, etc.

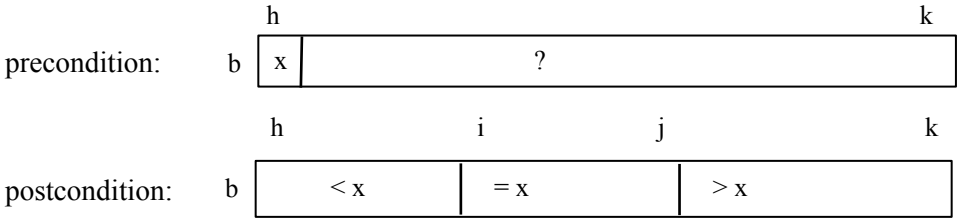
(a). Write the body of the constructor of class `MP3Song` on the bottom of page 1.

String function `s.endsWith(s1)` can be used to determine whether String `s` ends with String `s1`.

(b). Write a class `MP3` that extends subclass `HardDrive` of question 1. An instance is a small thumb drive that contains only mp3 songs. Each object on this drive should be an object of class `MP3Song` , giving the song and its title (see page 1). Provide the following methods. Do not add fields. There is no need to specify these methods since we give the specs here.

- A constructor `MP3(int c)`, where `c` is its capacity.
- `add(Object mp, String n, int size)`: A function that adds song `mp` with title `n` to this drive and returns its location. The size of song `mp` is `s`. Throw an `IllegalArgumentException` if the title does not end in ".mp3".
- `getTitle(int i)`: A function that returns the title of the song at location `i` of this drive.
- `getSong(int i)`: A function that returns the actual song at location `i` of this drive.

Question 3 (13 points). Algorithms. We want to use quicksort on arrays that have many equal elements. In order to speed up the algorithm, we use a slightly different partition algorithm. Its precondition and postcondition appear below, as well as a header for method partition. Note that i and j should be declared as local variables of the method. (1) Draw the invariant, as a diagram, in the place shown. After this, do *not* write on the invariant at all. If you have to, draw it to the side and then scratch on it. (2) Write the body of the method, making sure to use the invariant you wrote. Don't forget the return statement. Class Point is given below.



invariant:

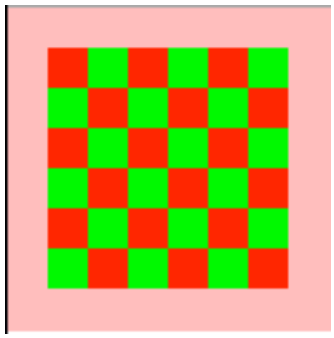
/** Given array segment b[h..k] as shown above, with h <= k, where x denotes the value initially in b[h], swap the values of b[h..k] and store in variables i and j so that the postcondition shown above is true; then return the Point (i, j). */
public static Point partition(**int**[] b, **int** h, **int** k) {

}

```
/** An instance is an immutable point (x,y) */
public class Point {
   public final int x;   public final int y;

   /** Constructor: An instance for (x, y). */
   public Point(int x, int y) {
      this.x= x;  this.y= y;
   }
}
```

Question 4 (13 points) Two-dimensional arrays. Recall drawing bricks, or rectangles, in A7. Write method `placeSquares`, whose specification and header appears below. It draws square bricks as shown to the right.



Here are the important points.

• There are m columns and rows of squares; precondition: 0 < `m`.

• Each square has side length `BrickSide`, and there is no space between them.

• The top-left square is at the upper-left corner (0,0) of the GUI.

• Squares in columns and rows 0 and `m-1` have color `Color.pink`;

• Inner squares have a checkerboard pattern of `Color.red` and `Color.green`, as shown to the right (the upper-left one is red; the one next to it, green).

• Class `Brick`, which extends `GRect`, has this additional constructor:

```
/** Constructor: a new square brick at (x,y) with side length s.*/
public Brick(double x, double y, double s)
```

• Class `Brick` has this method:

```
/** Give the brick color c */
public void setColor(Color c)
```

• Add a `Brick b` to the GUI using `add(b);` .

```
/** Create an array of m x m squares (Bricks), as specified above, also adding the squares to the GUI,
    and return the array. */
public Brick[][] placeSquares(int m) {




}
```

Question 5 (14 points). Recursion and loops with invariants.

(a) Complete the body of the following function —the repetend and all the underlined places. Note that the invariant is given and must be used, so study it carefully. Examples: eq("aaaxxyx", 0) is 3 and eq("aaaxxyx", 5) is 1. Be careful with the loop condition; there are two situations in which execution of the loop should terminate.

```
/** = the length of the sequence of equal characters beginning at s[i].
      Precondition: 0 <= i < s.length(). */
  public static int eqChars(String s, int i) {
    int k= _____;
    /** invariant: Characters in s[i..i+k-1] are the same */
    while (_____) {



    }
    return _____;
  }
```

(b) We want to compress strings that have long sequences of equal characters. For example, we want to compress "bbbbaaa$$$$$$$$$$$$$$$$d" to "b4a3$16d1". In the compression, each sequence of equal characters is given by the character followed by the length of the sequence. Write function compress to do this. Use no loops; use only recursion. You can use function eqChars.

```
/** = the compression of s, as explained above.
      Precondition: s contains no digits 0..9. */
public static String compress(String s) {









}
```

Question 6 (14 points) Miscellaneous.
(a) Below is an abstract class, whose only components are abstract methods. To the right, write this class as an interface.

```
public abstract class P {
  public abstract int m(int x);
  public abstract void p();
}
```

(b) Write a class that has `JFrame` as a superclass, that implements interface `P` of part (a), and whose only components (fields and methods) are those that are required. If you have to declare methods, you get to decide what they do.

(c) We have forgotten how to find the length of a string `s`, and we are in a hurry. We do remember that `s.charAt(k)` throws a `StringIndexOutOfBoundsException` if `k` is not the index of a character `s`. So we (meaning you) write the function below, using a loop (with initialization) that successively evaluates `s.charAt(0)`, `s.charAt(1)`, `s.charAt(2)`, ... until the exception is thrown, at which time `k` will be the length! Write the body of the function. You will need a try-statement.

```
/** = length of string s */
public static int length(String s) {




}
```

Question 7 (14 points) Executing sequences of statements. Some of these questions deal with the two classes at the bottom of this page.

(a) Explain how to execute an if-else statement `if (expression) S1 else S2`

(b) Evaluate the new-expression **new** Alpha(5). As you do it, draw any objects that are created and draw the frames for any method calls that are executed. Don't erase the frames; just cross them out. Write the value of the expression here: _____.

(c) Below are three statements. Draw the variables declared in the statements and then execute the sequence of statements, drawing any objects that are created. Do not draw frames for calls.

```
Alpha c= new Alpha(2);
Beta b= new Beta(3);
Alpha a= b;
```

```
public class Alpha {
   public int f;

   public Alpha (int n) {
     f= 2*n;
   }

   public int deal() {
     return f / 2;
   }
}
```

```
public class Beta extends Alpha {

   public Beta(int n) {
     super(2*n);
   }

   public int deal() {
     return f + 1;
   }

}
```