# CPS 4: Exam 2

<table>
<thead>
<tr>
<th>Possible</th>
<th>Earned</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>Loops</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Collections</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Colors</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Levels</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Searching and Sorting</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Debugging</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Asteroids: Bugs #1</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Asteroids: Bugs #2</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Asteroids: Features</td>
</tr>
<tr>
<td>+9</td>
<td></td>
<td>Extra Credit</td>
</tr>
</tbody>
</table>

100 Total
Loops

Currently, the code in Asteroids that adds the walls to the animation canvas looks like this:

```java
canvas.addSprite(boundaries[0]);
canvas.addSprite(boundaries[1]);
canvas.addSprite(boundaries[2]);
canvas.addSprite(boundaries[3]);
```

Write a loop below that does the same thing as the above code. (4 pts.)

```
for (int i = 0; i < boundaries.length; i++) {
    canvas.addSprite(boundaries[i]);
}
```

Label the **initialization**, **test (condition)**, **update**, and **body**, in your loop above. (4 pts.)
Say you want to add a special missile to the game that seeks and destroys the closest visible asteroid to the ship. Part of the feature is implemented for you below. All you need to do is finish writing a loop. After this loop, the variable “closest” should contain the index of the visible asteroid which is closest to the ship. This is “finding an extreme value in a collection;” and it is very similar to finding the car that is in the lead in Racers. (8 pts.)

```java
private void handleInput()
{
    // ... (a bunch of code here not shown)
    else if (key == 'm')
    {
        int closest = 0; // will end up being the index of the closest asteroid (when // the loop is done)
        double minDistance = 99999999; // start this out “infinitely” big; it will end // up being the distance to the closest asteroid
        Coordinate s = ship.getLocation(); // the ship's location

        // find the closest asteroid's index, with a loop over all asteroids
        for (int i = 0; i < asteroids.length; i++)
        {
            Coordinate a = asteroids[i].getLocation();
            double distance = a.distance(s);

            if (asteroids[i].isVisible() && distance < minDistance)
            {
                minDistance = distance;
                closest = i;
            }
        }

        // shoot the missile at the closest asteroid
        fireMissileAtAsteroid(asteroids[closest]);
    }
    // ... (a bunch of code here not shown)
}
```
What are two differences between arrays and ArrayLists? (2 pts. each; 4 pts. total)

array
- fixed size once created
- stores primitives or objects
- access/change elements in the array by indexing in brackets (e.g., `blocks[3] = new RectangleSprite(2,1), and blocks[3].setColor(new Color “Green”)`)

ArrayList
- dynamic size
- stores only objects, not primitives
- access/change elements using add and get methods (e.g., `blocks.add(new RectangleSprite(2,1)), and blocks.get(3)`)

Below, declare and create a collection (array or ArrayList) for 10 Color objects. Call it “grays.” You do not have to create the individual Color objects within the array (with a loop) quite yet (note that this means if you choose an ArrayList, it doesn’t matter that I said the number of objects that the collection will store…). This should only take one or two lines of code. (4 pts.)

```java
Color[] grays;
grays = new Color[10];
```

or,

```java
ArrayList<Color> grays;
grays = new ArrayList<Color>();
```
Create all of the 10 individual colors in the collection called “grays” that you created in the last question. As the index in the array increases, the shade of gray should get lighter (e.g., grays[0] is darker than grays[1]). To get full credit, you must use a loop. Recall that the Color constructor takes three integer arguments, one for red, one for green, and one for blue. (6 pts.)

```java
for (int i = 0; i < grays.length; i++)
{
    grays[i] = new Color(20 * i, 20 * i, 20 * i);
}
```

Say you’re working on Breakout, and you’re creating the blocks with a loop. The line below sets the color of each block. It is within a for loop, where i ranges over all the valid indices for the array of blocks:

```java
blocks[i].setColor(new Color("Gray"));
```

Write a new line of code below (replacing the line above) which will set the color of each block to a different one of the Color objects in your collection “grays,” instead of setting all blocks’ colors to the same color (as the line above would do). Note: To get full credit, you must ensure that the program won’t crash if the number of blocks is greater than the number of colors in “grays”. Also note: The line you’re writing is already within a for loop, so you don’t need to write another loop. (4 pts.)

```java
blocks[i].setColor(grays[i % grays.length]);
```

Write different replacement code below, which instead sets the color of each block to a new Color object which is a random shade of gray (it doesn’t use the “grays” array at all). Remember that “random.nextInt(m)” gives you a random integer between 0 and m (not including m). Note: You need to declare and initialize an int variable, to be used in creating your color…this is started for you. (3 pts.)

```java
int x = random.nextInt(256);
blocks[i].setColor(new Color(x, x, x));
```
Currently, what two aspects of game play in Asteroids change with each new level (other than simply displaying a different number X in “Level: X”)? (2 pts. each; 4 pts. total)

(1) more asteroids

(2) different score earned for shooting the asteroids

Say we wanted to make the asteroids grow in size as you progress through levels (e.g., the asteroids in level 2 are bigger than the asteroids in level 1, etc.). We can change one line in Asteroids.java to do this. Find the line. Write what method it is in, and then write the changed line. (4 pts.)

The line is in initAsteroids.

asteroids[k].setScale(ASTEROID_SCALE * level);

In “changeLevels()”, when we move on to the next level we remove all sprites, and then call “initSprites()” to recreate them and position them in their initial locations. Why don’t we call “startGame()” instead to reinitialize everything -- what would go wrong? (3 pts.)

If we called startGame(), we would reset all our sprites (which we want), but we would also set the score to be 0, the level to be 1, and the lives to be NUM_LIVES, which we don’t want -- the we don’t want to reset these things when we change levels!
Can you do binary search on *unsorted* lists? (2 pts.)

No.

What search algorithm(s) can you do on an *unsorted* list? (2 pts.)

**Linear search.**

Write out the binary search algorithm in English or Java. *Try to be very “step by step,” so a computer (that understood English) could follow your instructions.* (6 pts.)

1. Look at the thing in the middle of the list (or, if the list has an even # of things, the thing just to the right or left of the middle).
2. If this is the thing you’re looking for, you’re done!
3. Otherwise, check if the thing you’re looking for is before or after this middle thing (this could be alphabetically, or numerically -- however the list is sorted). If it’s after, throw away everything *before* the middle thing; if it’s before, throw away everything *after.*
4. Repeat from step 1 (in the part of the list you didn’t throw away, of course).

If you search for something in a list of length 64 using the **linear search** algorithm, the searching will require at maximum 64 comparisons before you find what you were looking for; and if we doubled the list size to 128, we would need a maximum of 128 comparisons. Therefore, the linear search algorithm takes O(N), or linear, time. In other words, the time needed is directly proportional to the length of the list. On the other hand, if you searched for something in a list of size 64 using **binary search**, it would take only 6 comparisons at maximum; and if you doubled the list size to 128, you would only need 7 comparisons at maximum. How would you describe the time requirement of binary search (using “O (something),” or “proportional to…”)? (3 pts.)

O(log (N)), or proportional to the log of the length of the list.

What’s one *slow*, but simple, sorting algorithm (just the name is fine)? (2 pts.)

**Selection Sort.** (and a fast one is **QuickSort**)

*Extra Credit:* If there are no collisions, what is the time needed to search for an item in a hash table, in O(something) notation? (+2 pts.)  O(1) -- constant time (doesn’t change based on the list length)
1. What type of error is almost always signified by a red squiggly line underneath code in Eclipse (and a lightbulb with an X in it on the left sidebar)? There are two acceptable answers. (3 pts., +1 extra credit for both… but only 1 point given total if the second one is wrong)

**a syntax error, a.k.a. a compilation error**

2. What type of error is signified by a difference between what you want your code to do and what you think it should do (a “bug” in game play, such as “lives” decreasing below zero, or a block not disappearing when the ball hits it, etc.)? There are two acceptable answers. (3 pts., +1 extra credit for both… but only 1 point given total if the second one is wrong)

**a semantic error, a.k.a. a runtime error**

3. If we changed line 55 in Asteroids.java from this:

   ```java
   int lives;
   ```

   to this:

   ```java
   lives;
   ```

   would that cause the type of error in question 1 or the type in question 2? (2 pts.)

**question 1**

4. Line 79 in Asteroids.java is currently this:

   ```java
   lives = NUM_LIVES;
   ```

   If we completely remove this line, would that cause the type of error in question 1 or the type in question 2? (2 pts.)

**question 2**
The Asteroids game currently has the following problem: If the ship is in (or very near) its starting position, and it is destroyed by an asteroid, it will be immediately hit again by the same asteroid when it is recreated for its next life. Thus, all lives can be used up due to multiple nearly-instantaneous collisions with one asteroid. There are many possible fixes for this. One simple fix is to destroy the asteroid, as well as the ship, when a ship-asteroid collision occurs. Add to the code shown below to implement this fix. Note: I know this isn’t the best fix -- instead, most asteroids games give you a couple seconds of invincibility when your ship is recreated... (3 pts.)

```java
if (ship.intersects(rock)) {
    // puts the ship back in the center with zero velocity
    initShip();

    lives = lives - 1;

    // destroy the asteroid
    rock.setVisible(false);
}
```

The Asteroids game has another problem: The ship can fly infinitely fast, but we want its speed limited. There is already a constant in the ShipTracker called MAX_SPEED, but it is never used. Add to the code below to limit the speed using this constant. (hint: all you need is an if statement) (5 pts., +2 extra credit if your code also limits the speed when the ship is going backwards)

```java
public void thrust (double amount) {
    double speed = velocity.getSpeed();

    if (Math.abs(speed + amount) < MAX_SPEED) {
        velocity.accelerate(amount);
    }

    if ((speed < MAX_SPEED && amount > 0) ||
        (speed > -MAX_SPEED && amount < 0)) {
        velocity.accelerate(amount);
    }
}
```
A bug has been sneakily introduced into the printed version of the Asteroids code (which wasn’t there in the snarfable Asteroids code). **The bug is in “handleCollisions.”**

Here’s the effect of the bug: We want our score to increase every time we hit an asteroid, but instead, something else happens.

Perhaps it’s easy to find the bug, but regardless, tracing how the “score” and “level” variables change as we shoot asteroids should make things clear.

Write the values for these two variables after each asteroid is destroyed. Here are some helpful constants from the game for reference:

```java
final int NUM_ASTEROIDS = 3; // # asteroids on 1st level
final int ASTEROID_POINTS = 100; // # points per asteroid on 1st level
```

(5 pts.)

<table>
<thead>
<tr>
<th></th>
<th>score</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>initially</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1st asteroid destroyed</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2nd asteroid destroyed</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>3rd asteroid destroyed</td>
<td>100</td>
<td>1 or 2</td>
</tr>
<tr>
<td>4th asteroid destroyed</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>5th asteroid destroyed</td>
<td>200</td>
<td>2</td>
</tr>
</tbody>
</table>

Find the line with the bug in it. It is in the “handleCollisions” method. Fix the bug, and write the **changed** line below. (3 pts.)

```
score = score + level * ASTEROID_POINTS;
```
Some versions of Asteroids allow the player to turn on a **shield**, which protected it from collisions with asteroids for a short time. Let’s say we’ve declared the following instance variable in the Asteroids class:

```java
/** the graphical representation of the shield */
Sprite shield;
```

We’ve also added the following code to “initSprites()” to create the shield, and initialize its characteristics:

```java
// initialize the shield
shield = new EllipseSprite(1,1);
shield.setScale(SHIP_SCALE);
shield.setLocation(ship.getLocation());
shield.setColor(SHIP_COLOR);
shield.setTracker(shipTracker); // so it is always in the same position as the ship
shield.setVisible(false); // make the shield be invisible initially
canvas.addSprite(shield);
```

When the shield is on, we want the asteroids to **pass harmlessly through the ship** (but they are not destroyed). Also, when the shield’s on, the **player cannot fire any bullets** (so you can’t just leave it on all the time and shoot asteroids with impunity).

Describe how to change the following two methods of the Asteroids class to implement this feature such that the shield is turned on when the player presses the ‘s’ key. You can use English rather than code, but describe the logic in such detail that a Java programmer could immediately and easily write the code from your instructions, given a computer and Eclipse (i.e., don’t say “turn the shield on,” but instead say “make the shield visible,” or “shield.setVisible(true”)”). **Note: Assume the shield turns off automatically** (since figuring out when to turn it off is a timer issue). (5 pts. per method)

**handleInput:**

*If the player presses ‘s’, make the shield visible.*

Add the condition that the shield is *not* visible to the *if* statement for firing bullets (i.e., if they press ‘f’ and the bullet is not visible and the shield is not visible, then initBullet()).

**handleCollisions:**

*Add the condition that the shield is not visible to the if statement that currently runs if the ship intersects an asteroid (i.e., if ship.intersects(rock) and shield.isVisible() == false, then reset the ship, and subtract a life).*
Extra Credit: Useless Facts

1. Name one useless fact of your own choosing. It may not be “common knowledge” (for example, “the sky is blue” would not count). I will decide whether or not your fact counts. **Note:** Since I said in the study guide that the *Useless Facts of the Day* (from class) would not be on the test, you may not name one of these. **Also note:** **It cannot be the fact you gave on the last test!** (1 pt. for something bordering on common knowledge, 2 pts. for something weird)

2. What is the most common last name in the world? (0.5 pts.)

   **Chang, Zhang, Li, or Wang (about.com and Wikipedia)**
   
   There are about 100 million people in China with each surname (each is about 7.5% of the Chinese population)!

3. What is the most common last name in the U.S.? (0.5 pts.)

   **Smith (last name list from 1990 U.S. census)**
   
   There are about 2.5 million Smiths in the U.S. (only 1.01% of the U.S. population).