Active Objects

Name: _____ ID: _____

These questions do not have a formal, definitive answer. They are meant to be food for thoughts. Feel free to seek answers on browsing the Internet, talking to other software developers or reading books.

1. What are the differences between processes and threads?

2. What is an active object?

3. There are two ways to instantiate a thread in Java. Either we implement `Runnable` or we extend `Thread`. Illustrate each approach with an example.

4. What does the program below do? Why is it necessary to catch an exception?

```
public class SleepMessages {
  //Display a message, preceded by the name of the current thread
  static void threadMessage(String message) {
    String threadName = Thread.currentThread().getName();
    System.out.format("%s: %s%n", threadName, message);
  }
  public static void main(String args[]) throws InterruptedException {
```

```
    String importantInfo[] = {
      "Twinkle Twinkle Little Star",
      "How I wonder what you are",
      "Up above the world so high",
      "like a diamond in the sky"
    };
    for (int i = 0; i < importantInfo.length; i++) {
      Thread.sleep(2000);
      threadMessage(importantInfo[i]);
    }
  }
}
```

5. An instance of `Thread` has an `interrupt()` method that can be invoked on that object to stop it. What is the effect of invoking `interrupt()` on an object?

6. What does the program below do? Add comments to the code, explaining what each important part does. This program has a number of special methods, defined in the `Thread` class. Explain the purpose of these methods.

```
public class SimpleThreads {
  public static void main(String args[]) throws InterruptedException {
    long patience = 1000 * 60 * 60;
    if (args.length > 0) {
        patience = Long.parseLong(args[0]) * 1000;
    }
    MessageLoop.threadMessage("Starting MessageLoop thread");
    long startTime = System.currentTimeMillis();
    Thread t = new Thread(new MessageLoop());
    t.start();
    MessageLoop.threadMessage("Waiting for MessageLoop thread to finish");
    while (t.isAlive()) {
      MessageLoop.threadMessage("Still waiting...");
      t.join(1000);
      if (((System.currentTimeMillis() - startTime) > patience) &&
          t.isAlive()) {
        MessageLoop.threadMessage("Tired of waiting!");
```

```
        t.interrupt();
        t.join();
      }
    }
    MessageLoop.threadMessage("Finally!");
  }
}
```

7. What problems might happen if two threads invoke methods on instances of the following class? Write a modified version of this class that solves this problem.

```
class Counter {
    private int c = 0;
    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int value() {
        return c;
    }
}
```

8. In Java, we cannot mark constructor method with the `synchronized` keyword. Why synchronized constructors do not make sense?

9. If an object has two fields that are never used together, then synchronizing their access may be a waste of time. Rewrite the program below to synchronize the updates of the two fields, knowing that they are never accessed at the same time.

```
public class MsLunch {
```

```
    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();
    public void inc1() {
        synchronized(lock1) {
            c1++;
        }
    }
    public void inc2() {
        synchronized(lock2) {
            c2++;
        }
    }
}
```

10. The program below has a problem known as *deadlock*. Explain what is this, and why this happens the given program:

```
public class Deadlock {
  static class Friend {
    private final String name;
    public Friend(String name) {
      this.name = name;
    }
    public String getName() {
      return this.name;
    }
    public synchronized void bow(Friend bower) {
      System.out.format("%s: %s has bowed to me!%n",
          this.name, bower.getName());
      bower.bowBack(this);
    }
    public synchronized void bowBack(Friend bower) {
      System.out.format("%s: %s has bowed back to me!%n",
          this.name, bower.getName());
    }
  }
  public static void main(String[] args) {
```

```
    final Friend alphonse = new Friend("Alphonse");
    final Friend gaston = new Friend("Gaston");
    new Thread(new Runnable() {
        public void run() { alphonse.bow(gaston); }
        }).start();
    new Thread(new Runnable() {
        public void run() { gaston.bow(alphonse); }
        }).start();
  }
}
```

11. Two other common concurrency hazards are *starvation* and *livelocks*. Explain each of these concepts.

12. The program below uses a shared variable `joy` to wait for an event. It has a problem. Explain this problem, and show how to fix it.

```
public void guardedJoy() {
  //Simple loop guard. Wastes processor time. Don't do this!
  while(!joy) {}
  System.out.println("Joy has been achieved!");
}
```

13. *Data races* are very common problems in concurrent programming. Explain what is a data race.

14. A way to avoid data races is by using *immutable objects*. Explain what is an immutable object, and then show how to change the class below so that its instances are immutable:

```
public class SynchronizedRGB {
  //Values must be between 0 and 255.
  private int red;
  private int green;
  private int blue;
  private String name;
  public SynchronizedRGB(int red, int green, int blue, String name) {...}
  public void set(int red, int green, int blue, String name) {...}
  public synchronized int getRGB() {...}
  public synchronized String getName() {...}
  public synchronized void invert() {...}
}
```