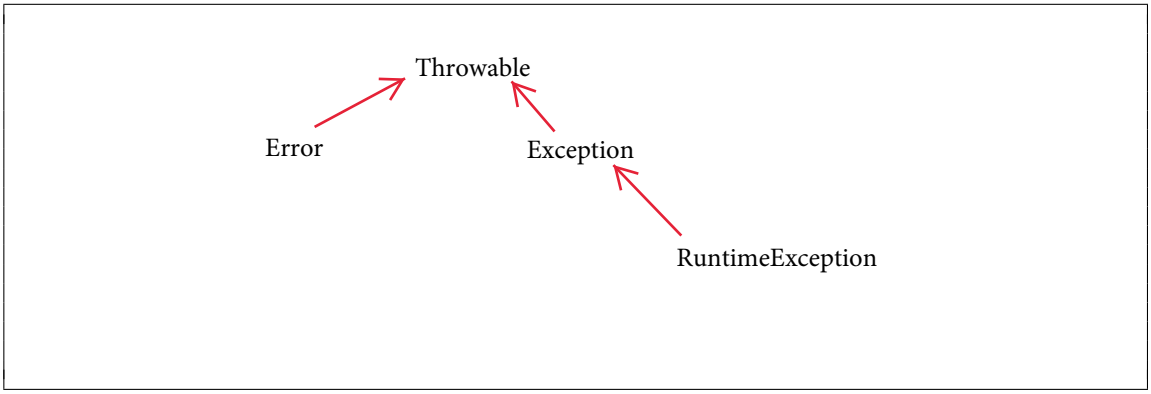
This homework is individual. Total points is 15.

1. Exceptions.

   (a) (3 points) Draw the type hierarchy (the subclass/superclass relation) of the four Java class/interface: `Throwable, Exception, RuntimeException, Error`. An arrow from a class B to class A means class B is a subclass of class A.



   (b) (2 points) What is the typical usage of *Checked Exceptions*? Given an example.

Checked exceptions are used for recoverable conditions, where the caller can be expected to recover. An example is FileNotFoundException.

   (c) (2 points) What is the typical usage of *Unchecked Exceptions*? Given an example.

Unchecked exceptions are used to indicate programming errors. An example is ArrayOutOfBoundsException

   (d) (2 points) What is the typical usage of *Errors*? Given an example.

Errors are reserved for use by the JVM to indicate resource deficiencies, invariant failure, or other things that stop the program. An example would be StackOverflowError.

2. (3 points) As we learned in class that it is not a good design showing the user of the program that we are using plain `int` to store student ids. The same idea applies to representing temperatures. Design a Java class `Kelvin` which wraps an `int` value of the actual temperature in kelvin unit. Use the getter and setter pattern to maintain the integrity of this temperature value. Throw a runtime exception when the user wants to set an invalid value.

```
public class Kelvin {
  private int value;
  public int get() {
    return value;
  }
  public void set(int newValue) {
    assert newValue > 0;
    value = newValue;
  }
}
```

3. (3 points) In many cases the getter-setter pattern cannot guarantee the integrity of the data. Consider the following code:

```
class Student {
  private List<Course> courses = new ArrayList<>();
  public List<Course> getCourses() { return courses; }
  public void addCourse(Course another) {
    assert isValid(another);
    courses.add(another);
  }
}
```

Show the code to add an invalid course to a student `s`.

```
List<Course> lst = s.getCourses();
lst.add(invalidCourse);
```