Problem 1: Polynomial Factory

We want to have a tool for evaluating polynomials and computing their roots.

(1) Write an interface called Polynomial that has two functions:

- Evaluates a polynomial (input is a double, output is a double)
- Evaluates the roots of a polynomial (no input, outputs an array of double)

(2) Implement this interface for linear polynomials. Note that a linear polynomials store two values a, b, are evaluated by $f(x) = ax + b$, and their root is defined by $-b/a$.

(3) Implement this interface for quadratic polynomials. Note that a linear polynomials store two values a, b, c, are evaluated by $f(x) = ax^2 + bx + c$, and their root is defined by $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Note that $\sqrt{\cdot}$ is found at java.lang.Math.sqrt.

(4) Implement a class called PolynomialFactory that creates a polynomial. You should use switch/case. It implements this interface:

```
interface PolynomialFactoryInterface {
    // if two coefficients are passed in it returns a LinearPolynomial
    // if three coefficients are passed it returns a QuadraticPolynomial
    // otherwise it returns null
    Polynomial getPolynomial(double[] coefficients);
```

(5) re-implement getPolynomial using if/else.

(6) Implement a function that takes in a PolynomialFactory and a two dimensional array of polynomial coefficients (an array of double[]) and constructs an array of Polynomials.

(7) Implement a function meanPolynomialValue that takes in an array of Polynomials and a double t and returns the average value of the polynomials passed in.

```
Polynomial[] polynomials =getPolynomials(...)
double t = 34.0;
double meanValue = meanPolynomialValue(polynomials,t);
```

(8) Implement a function with the same prototype but returns the min value minPolynomialValue. Take care on how you initialize the minimum variable.

(9) Implement a function that computes the average root meanRoot.

```
Polynomial[] polynomials =getPolynomials(...)
double meanRoot = meanRoot(polynomials);
```

Problem 2: Forward Euler canon

In this problem we will simulate the path of a two-dimensional canon ball shot on a flat plane. The goal is to capture when the canon ball lands.

(1) Implement a class that represents a Ball's state. For this problem we need to store the x and y coordinate of the ball's position and its current velocity (which also has x and y components). It should follow the following interface:

```
interface EulerInterface {
    // simulates forward by the specified amount of time using Forward Euler.
    boolean step(double dt);
    // returns whether the ball is in the ground.
    // This should be when the y coordinate is at most $0$.
    boolean inGround();
}
```

Forward Euler updates the state of the ball by the following equations:

$$p = p + (dt)(v) \tag{1}$$
$$v_y = v_y - 9.8. \tag{2}$$

(2) implement a function that moves a canonball forward until it hits the ground.

```
double findCollision(CanonBall ball) {




}
```

Problem 3: Pigeonhole Principle

The pigeonhole principle states that if we have n pigeons in m holes, then if we have more pigeons than holes then at least one hole has more than one pigeon. In this problem we will help pigeons find holes.

```
public class Pigeon {
    private String name;
    Pigeon(String name) { this.name = name; }
    String getName() { return name; }
}
```

(1) Create a checked exception called CannotPlacePigeonException that stores the Pigeon that could not be placed. If this error is printed it should show a message using the pigeon's name.

```
interface PigeonHolesInterface {

    // returns the number of holes available
    int size();

    // returns true if there is a pigeon in hole index
    boolean hasPigeon(int index);

    // places a pigeon in a particular hole
    void placePigeon(Pigeon pigeon, int index) throws CannotPlacePigeonException;

}
```

(2) Implement a function that tries to place a pigeon in a hole. Assume all of the pigeons passed in had not been previously placed. Should this method throw an exception?

```
public static void placePigeon(PigeonHolesInterface holes, Pigeon pigeon) {









}
```

(3) Implement a function that tries to stuff as many pigeons into holes as possible. Assume all of the pigeons passed in had not been previously placed. Should this method throw an exception? You may use the previous placePigeon.

```
public static void stuffPigeons(PigeonHolesInterface holes, Pigeon[] pigeons) {









}
```

(4) Create a class called SinglePigeonHoles that represents a set of holes that pigeons can be placed in and satisfies the following interface:

```
interface SinglePigeonHolesInterface extends PigeonHolesInterface {
    // returns the pigeon in the hole index.
    // can return null if no pigeon is there
    Pigeon getPigeonInHole(int index);

    // count the number of holes that are empty
    int remainingHoleCount();
}
```

(5) Implement a function that prints out the contents of every hole, either printing the pigeon's name or saying "Hole is empty":

```
public static void printContents(SinglePigeonHolesInterface holes) {
```

```
}
```

(6) Write a class that implements SinglePigeonHolesInterface that uses a standard array of Pigeon to represent the pigeons. Note that if an entry in the array holds the value null then it does not have a pigeon in it.

(7) Write a class that implements PigeonHolesInterface where each hole can hold a number of holes by using a two dimensional array of Pigeon to represent the pigeons. The constructor should take in two arguments: the number of holes and the number of pigeons that can fit in each hole.