6.1 You have 20 bottles of pills. 19 bottles have 1.0 gram pills, but one has pills of weight 1.1 grams. Given a scale that provides an accurate measurement, how would you find the heavy bottle? You can only use the scale once.

Solutions:
Take one pill from Bottle #1, two pills from Bottle #2, three pills from Bottle #3, etc. Weigh these pills together. If all the pills were one gram each, then the scale would read 210 grams (1 + 2 + 3 + 4 + ... + 20 = 210) or (20 * 21)/2 = 210. Any weight over 210 must come from the heavier pills.

Then use this formula to find the bottle number. ((weight - 210) / 0.1).
So for example, if the set of pills weighed 211.3 grams, then bottle #13 would have the 1.1g pills.

6.2
(1) To win Game 1 : The probability is p1 = p
(1) To win Game 2 : The probability is p2 = p^3 + 3*(1-p)*p*p = 3*p^2 - 2 * p^3

if p1 > p2 , so that 0<p< 0.5, we should choose Game1
if p = 0 or 0.5 or 1, then p1 == p2, Game1 and Game2 have the same probability of winning.
if 0.5 < p < 1, then p1 < p2, we should choose Game2

6.3 Dominos:
Solutions:
No, we cannot cover.
Proof: since the two diagonally cut-off corners are of the same color, without loss of generality, we can assume they are both black, so the left board has 30 black cells and 32 white cells. However, for each domino, it covers 1 black cell and 1 white cell, so in total they need 31 black cells and 31 white cells. And it is impossible for us to cover 30 black cells and 32 white cells with these dominos.

6.4
Each ant has two possible directions, and if they all go in the same direction (clockwise or counterclockwise), there will not be any collisions.
And the collision probability is 1- P(same direction) = 1- (1/2)^3 - (1/2)^3 = 3/4
Similar for n-vertex polygon, the collision probability = 1- 2* (1/2)^n = 1 - (1/2)^(n-1)

6.5

Step#	amount of water in 5-quart jug	amount of water in 5-quart jug
1	5	0
2	2	3
3	0	2
4	5	2
5	4	3

Steps are detailed as follows:

1. Fufill the 5-quart jug.
2. fullfill the 3-quart jug with the water in 5-quart jug.
3. Empty the 3-quart jug, then pour all the left water(2 quarts) in 5-quart jug into the 3-quart jug.
4. Fufill the 5-quart jug.
5. pour water from 5-quart jug into 3-quart jug until the 3-quart jug is full.

And now , there is 4 quart water in the 5-quart jug.

6.6

Solutions: if there are c people with blue eyes, it needs c nights for all blue-eyes people to leave the island, and they will all leave at the c-th night together,

First assume there are n people on this island, and c people are with blue eyes. $c>0$

And we analyse 3 scenarios:

(1): $c=1$

So the only one person with blue eyes can find that no other people have blue eyes, but he knows that there is at least one person with blue eyes, so he can deduce that he himself is with blue eyes. And in the end, he will take the plane to leave at the first night. So it only takes 1 day.

(2): c = 2

The two people (we name the A, B) see each other , but they are not sure whether c is 1 or 2. And they both know that if c equals 1, the blue-eyes guy will leave at the first night. So they both will stay to see and will not leave at the first night. And on the second day, they will find out must be 2 and he himself is with blue eyes. So these two blue-eye people will both leave at the second day's night. So it takes 2 days.

(3): c > 2

Similarly, if c =3, the 3 blue-eye people will think that 2 or 3(if himself included) are with blue eyes, and if two nights passed but no one left, then that guy can realize he himself is blue-eyes. So there 3 people will all leave at the third night.

So whatever c is , we analyze similarly and the result is: if there are c people with blue eyes, it needs c nights for all blue-eyes people to leave the island, and they will all leave at the c-th night together,

6.7

/* The Apocalypse*/

The probability can be the sum of (i/ (2^i)), where i goes from 0 to infinity, and this will be 1.

This means for each family, they will have exactly one girl and on average one boy.

So the gender ratio is even fro next generation.

For simulation:

```
package Q6_07_The_Apocalypse;

import java.util.Random;

public class Question {
	public static int[] aFamily() {
		Random random = new Random();
		int boys = 0;
		int girls = 0;
```

```
            while (girls == 0) {
                if (random.nextBoolean()) {
                    girls += 1;
                } else {
                    boys += 1;
                }
            }
            int[] genders = {girls, boys};
            return genders;
        }

        public static double runNFamilies(int n) {
            int boysTotal = 0;
            int girlsTotal = 0;
            for (int i = 0; i < n; i++) {
                int[] genders = aFamily();
                girlsTotal += genders[0];
                boysTotal += genders[1];
            }
            return girlsTotal / (double) (boysTotal + girlsTotal);
        }

        public static void main(String[] args) {
            double ratio = runNFamilies(10000000);
            System.out.println(ratio);

        }

}
```

6.8

No matter how we drop Egg 1, for Egg 2 we must do a linear search - from lowest bound to highest bound between the "breaking floor" and the next highest non-breaking floor. And to create a "load-balanced" system, we should get Drop(egg1)+Drop(egg2) always be the same.

1. Each drop of egg1 takes one more step, so egg2 is allowed to reduce one potential step

2. So if egg1 start at floor x, then it should go up x-1, then x-2 ...
3. solve x+(x-1)+(x-2)+...+1 = 100; => x=14

That is to say, the key in the problem is " worst case balancing “, and we drop from level 14, then tried level 27, 39, 50, 60,69, 77, 84, 90,95,99, …

6.9
Only 1, 2^2, 3^2, 4^2, ... 10^2 are open.
-- The number with odd factors
So in the end, only 10 lockers are open.

6.10
/* Poison: */
Seven days should be enough to solve the problem.

We can represent each bottle number in its binary form. And if the ith bit is 1, we add a drop of the bottle’s contents to stripe i. Since 1000 < 2^10 , this should be enough.
After waiting for seven days, we take back the stripes and interpret the result: if stripe i is negative, set the with bit of result value to be 1. In the end, from the binary-representation form of result value, we can know its decimal value, that is exactly the poisoned bottle number.

Simulations:

```
/*in file Bottle.java*/
public class Bottle {
	private boolean poisoned = false;
	private int id;

	public Bottle(int id) {
		this.id = id;
	}

	public int getId() {
		return id;
	}

	public void setAsPoisoned() {
		poisoned = true;
	}

	public boolean isPoisoned() {
		return poisoned;
```

```
	}
}



/*in file TestStrip.java*/

import java.util.ArrayList;

public class TestStrip {
	public static int DAYS_FOR_RESULT = 7;
	private ArrayList<ArrayList<Bottle>> dropsByDay = new
ArrayList<ArrayList<Bottle>>();
	private int id;

	public TestStrip(int id) {
		this.id = id;
	}

	public int getId() {
		return id;
	}

	/* Resize list of days/drops to be large enough. */
	private void sizeDropsForDay(int day) {
		while (dropsByDay.size() <= day) {
			dropsByDay.add(new ArrayList<Bottle>());
		}
	}

	/* Add drop from bottle on specific day. */
	public void addDropOnDay(int day, Bottle bottle) {
		sizeDropsForDay(day);
		ArrayList<Bottle> drops = dropsByDay.get(day);
		drops.add(bottle);
	}

	/* Checks if any of the bottles in the set are poisoned. */
	private boolean hasPoison(ArrayList<Bottle> bottles) {
		for (Bottle b : bottles) {
			if (b.isPoisoned()) {
				return true;
			}
```

```
            }
            return false;
        }

        /* Gets bottles that were used in the test DAYS_FOR_RESULT days ago. */
        public ArrayList<Bottle> getLastWeeksBottles(int day) {
            if (day < DAYS_FOR_RESULT) {
                return null;
            }
            return dropsByDay.get(day - DAYS_FOR_RESULT);
        }

        /* Checks if the test strip has had any poisoned bottles since before
DAYS_FOR_RESULT */
        public boolean isPositiveOnDay(int day) {
            int testDay = day - DAYS_FOR_RESULT;
            if (testDay < 0 || testDay >= dropsByDay.size()) {
                return false;
            }
            for (int d = 0; d <= testDay; d++) {
                ArrayList<Bottle> bottles = dropsByDay.get(d);
                if (hasPoison(bottles)) {
                    return true;
                }
            }
            return false;
        }
}


/*in file Solution.java*/
import java.util.ArrayList;
import java.util.Random;

public class Solution {
        public static ArrayList<Bottle> createBottles(int nBottles, int poisoned) {
            ArrayList<Bottle> bottles = new ArrayList<Bottle>();
            for (int i = 0; i < nBottles; i++) {
                bottles.add(new Bottle(i));
            }

            if (poisoned == -1) {
                Random random = new Random();
```

```
                poisoned = random.nextInt(nBottles);
            }
            bottles.get(poisoned).setAsPoisoned();

            System.out.println("Added poison to bottle " + poisoned);

            return bottles;
        }

        public static int findPoisonedBottle(ArrayList<Bottle> bottles, ArrayList<TestStrip>
strips) {
            runTests(bottles, strips);
            ArrayList<Integer> positive = getPositiveOnDay(strips, 7);
            return setBits(positive);
        }

        /* Add bottles to test strips */
        public static void runTests(ArrayList<Bottle> bottles, ArrayList<TestStrip> testStrips) {
            for (Bottle bottle : bottles) {
                int id = bottle.getId();
                int bitIndex = 0;
                while (id > 0) {
                    if ((id & 1) == 1) {
                        testStrips.get(bitIndex).addDropOnDay(0, bottle);
                    }
                    bitIndex++;
                    id >>= 1;
                }
            }
        }

        /* Get test strips that are positive on a particular day. */
        public static ArrayList<Integer> getPositiveOnDay(ArrayList<TestStrip> testStrips, int
day) {
            ArrayList<Integer> positive = new ArrayList<Integer>();
            for (TestStrip testStrip : testStrips) {
                int id = testStrip.getId();
                if (testStrip.isPositiveOnDay(day)) {
                    positive.add(id);
                }
            }
            return positive;
        }
```

```
	/* Create number by setting bits with indices specified in positive. */
	public static int setBits(ArrayList<Integer> positive) {
		int id = 0;
		for (Integer bitIndex : positive) {
			id |= 1 << bitIndex;
		}
		return id;
	}

	public static ArrayList<TestStrip> createTestStrips(int nTestStrips) {
		ArrayList<TestStrip> testStrips = new ArrayList<TestStrip>();
		for (int i = 0; i < nTestStrips; i++) {
			testStrips.add(new TestStrip(i));
		}
		return testStrips;
	}

	public static void main(String[] args) {
		int nBottles = 1000;
		int nTestStrips = 10;
		for (int poisoned = 0; poisoned < nBottles; poisoned++) {
			ArrayList<Bottle> bottles = createBottles(nBottles, poisoned);
			ArrayList<TestStrip> testStrips = createTestStrips(nTestStrips);
			int poisonedId = findPoisonedBottle(bottles, testStrips);
			System.out.println(“we think the poisoned Bottle is : " + poisonedId);
			if (poisonedId != poisoned) {
				System.out.println(“oops, wrong prediction");
				break;
			}
		}
	}
}
```