Chapter 6 - Math and Logic Puzzles

// **Prime Numbers**

- every positive integer can be decomposed into a product of primes
- gcd(x, y) * lcm(x, y) = x * y
- checking for primality

// Naive implementation checking if a number is prime

```
boolean primeNaive(int n) {
	if (n < 2) {
		return false;
	}
	for (int i = 2; i < n; i++) {
		if (n % i == 0) {
			return false;
		}
	}
	return true;
}
```

// A small improvement is to iterate only up through the square root of n

```
boolean primeSlightlyBetter(int n) {
	if (n < 2) {
		return false;
	}
	int sqrt = (int) Math.sqrt(n);
	for (int i = 2; i <= sqrt; i++) {
		if (n % i == 0) {
			return false;
		}
	}
	return true;
}
```

// In reality we only need to check if n is divisible by a prime number

// **The Sieve of Eratosthenes** → generate a list of primes → it works by recognizing that all non-prime numbers are divisible by a prime number

```
boolean[] sieveOfEratosthenes(int max) {
	boolean[] flags = new boolean[max + 1];
	int count = 0;

	init(flags);  // Set all flags to true other than 0 and 1;
	int prime = 2;

	while (prime <= Max.sqrt(max)) {
		crossOff(flags, prime);
		prime = getNextPrime(flags, prime);
	}

	return flags;
}

void crossOff(boolean[] flags, int prime) {
	for (int i = prime * prime; i < flags.length; i += prime) {
		flags[i] = false;
	}
}

int getNextPrime(boolean[] flags, int prime) {
	int next = prime + 1;
	while (next < flags.length && !flags[next]) {
		next++;
	}
	return next;
}
```

// **Probability of A and B** → two events A and B → overlapping area is the event {A and B} → what is the probability you would land in the intersection between A and B? → if you knew the odds of landing in A and the percent of A that's also in B (the odds of being in B given that you were in A) → P(A and B) = P(B given A) * P (A)

eg. What is the probability of picking an even number and a number between 1 and 5?

P (x is even and x <= 5) = P (x is even given x <= 5) * P(x <= 5)
= (2/5) * (5/10) = ⅕

→ since P(A and B) = P(B given A) * P(A) = P(A given B) * P(B) → P(A given B) = P(B given A) * P(A) / P(B) → Bayes' Theorem

// **Probability of A or B** → P(A or B) = P(A) + P(B) - P(A and B) eg. We pick a number between 1 and 10 (inclusive). What is the probability of picking an even number or a number between 1 and 5

P(x is even or x <= 5) = P(x is even) + P(x <= 5) - P(x is even and x <=5)
= 5/10 + 5/10 - ⅕ = ½ + ½ - ⅕ = ⅘

// **Independence** → P(A and B) = P(A) * P(B)

// **Mutual Exclusivity** → if A and B are mutually exclusive (if one happens then the other one can't happen) → P(A or B) = P(A) + P(B) → because P(A and B) = 0

// As long as two events have non-zero probabilities they will never be both mutually exclusive and independent

// If one or both events have a probability of 0 (impossible) then the events are both independent and mutually exclusive

// Ropes → You have two ropes and each takes exactly one hour to burn. How would you time 15 mins

x, y → time to burn ropes
x + y = 120
if we light a rope at both ends we can time 30 mins
x / 2 = 30
y - x/2

// **The Heavy Pill** → You have 20 bottles of pills, 19 have 1.0 gram pills and 1 has pills of 1.1 grams. You can only use the scale once

take 1 pill from bottle #1, 2 pills from bottle #2 … 20 pills from bottle #20

→ weight - 210 grams (if all pills weighed 1.0 grams) / 0.1 grams = # bottle

// **Basketball** → you can play 2 games: #1 → one shot to make the hoop, #2 → 3 shots to make 2 out of 3. Which game should you play?

P(#1) = p

s(k, n) = probability pf making k shots out of n

P(#2) = s(2, 3) + s(3, 3) → mutually exclusive

s(3, 3) = p^3 (probability of making one shot is 3) → independence of each shot

→ the probability of making exatly two shots is:
P(making 1 and 2, missing 3) + P(making 1 and 3, missing 2) + P(making 2 and 3, missing 1) =
p * p * (1 - p) * 3 = 3 * (1 - p) * p ^ 2

p = probability of making a shot → 1 - p = probability of mising the shot

→ P(winning #2) = p^3 + 3p^2 - 3p^3 = 3p^2 - 2p^3

you should play game 1 if P(#1) > P(#2)

p > 3p^2 - 2p^3 → solve inequality

// **Dominos** → 8 x 8 chessboard with 2 diagonally opposite corners cut off. Can you cover it with 31 dominoes considering 1 domino covers 2 squares?

chessboard has 64 squares → 32 black and 32 white
if we remove diagonally opposite corners we are removing 2 squares from the same color
→ let's say we have 30 B squares and 32 W squares
→ each domino we set on the board will take up 1 B square and 1 W square regardless how we place them
→ 31 dominoes will take up 31 B and 31 W → wo you can't cover the board

// **Ants on a Triangle** → find the probability of collision

→ the ants will collide if any of them are moving towards each other
→ they wont' collide if all of them are moving in the same direction (clockwise or counterclockwise)
→ P(clockwise) = ½ ^ 3 → ½ is the probability of one ant moving clockwise
→ P(counterclockwise) = ½ ^ 3
→ P(same direction) = P(C) + P(CC) = 2 * ½ ^3 = ¼
→ P(collision) = the probability of the ants not moving in the same direction
→ P(collision) = 1 - P(same direction) = 1 - ¼ = ¾
→ replace 3 (the number of vertices) with n and you can generalize the probabilities

// **Jugs of Water** → if the two jug sizes are relatively prime (or mutually prime → if the only positive integer that divides both of them is 1) you can measure any value between 1 and the sum of the jug sizes

// **Blue-Eyed Island**

→ assuming there are n people on the island and b of them have blue eyes → b > 0
→ case #1 → b = 1 → 1 evening as the blue eyed person can look around and see no-one has blue eyes therefore they must be the ones with blue eyes

→ case #2 → b = 2 → second night, the blue eyed people would see each other and assume that if b = 1 they would have left the first night
→ case #3 → b > 2 → using induction if b people have blue eyes it will take b nights (all people would leave same night)

// **The Apocalypse** → G indicates a girl and B indicates a boy

→ sequences of children will look like G, BG, BBG, BBBBBBG …
→ Work out the probabilities for each gender sequence
→ P(G) = ½ → P(BG) = ½ * ½ (independence)
→ P(BBG) = ½ * ½ * ½ = ⅛
→ how many boys does each family have on average?
→ the expected value of the number of boys is the probability of each sequence multiplied by the number of boys in that sequence
→ sum(i=0 to inf) = i/2^i → close to 1
→ gender ratio remains 50% girls 50% boys

```
// Simulation
public class Simulation {

	public static double runNFamilies(int n) {
		int boys = 0;
		int girls = 0;

		for (int i = 0; i < n; i++) {
			int[] genders = runOneFamily();
			girls += genders[0];
			boys += genders[1];
		}
		return girls / (double) (boys + girls);
	}

	public static int[] runOneFamily() {
		Random random = new Random();
		int boys = 0;
		int girls = 0;
		while (girls == 0) {
			if (random.nextBoolean()) {
				girls += 1;
			} else {
				boys += 1;
			}
		}
```

```java
        int[] genders = {girls, boys};
        return genders;
    }
}
```

// **The Egg Drop Problem**

```java
public class Question {

    private static int breakingPoint = 43;
    private static int countDrops = 0;

    public static boolean drop(int floor) {
        countDrops++;
        return floor >= breakingPoint;
    }

    public static int findBreakingPoint(int floors) {
        int interval = 14;
        int previousFloor = 0;
        int egg1 = interval;

        while (!drop(egg1) && egg1 <= floors) {
            interval -= 1;
            previousFloor = egg1;
            egg1 += interval;
        }

        int egg2 = previousFloor + 1;
        while (egg2 < egg1 && egg2 <= floors && !drop(egg2)) {
            egg2 += 1;
        }
        return egg2 > floors ? -1: egg2;
    }
}
```

// **100 Lockers** → a door n is toggled once for each factor of n including itself and 1
→ a door is left open if the number of factors (x) is odd
→ when would x be odd?
→ x is odd if n is a perfect square (if you pair factors the square root of a perfect square only contributes once → odd numbers of factors)
→ there are 10 perfect squares from 1 to 100

// **Poison** → naive approach (28 days)

→ divide the bottles across the 10 test strips first in groups of 100 wait 7 days and repeat for the group that came positive → 9 strips left 100/9 is the next size of groups (~12) → 8 strips → groups of 2 → 7 strips groups of 1 → 4 rounds * 7 days = 28 days

// Optimized approach (7 days) → each strip is a binary indicator for poisoned or unpoisoned

→ we can map 1000 keys to 10 binary values such that each key is mapped to a unique configuration of values