Q1:

No such file found

doing finally

-1

Q2:

a,

public class SolutionA {

public static void main(String[] args) {

SolutionA s = new SolutionA();

int result = s.kittenMittens(3);

System.out.println(result);

}

public int kittenMittens (int numKittens){

int count =0;

int sum = 0;

for (int i =1; i <= numKittens; i ++){

if (i%2 == 0){

count++;

}

}

int evenMeo = count \*3;

int oddMeo = (numKittens -count) \*4;

sum = evenMeo + oddMeo;

return sum;

}

}

b,

public class SolutionB {

public static void main(String[] args) {

SolutionB s = new SolutionB();

int res = s.kittenMittens(5);

System.out.println(res);

}

public static int kittenMittens(int numKittens){

int i = 0;

if(numKittens == 0) return i;

if(numKittens % 2 == 0) i = 3;

if(numKittens % 2 == 1) i = 4;

return kittenMittens(numKittens - 1) + i;

}

}

Q3:

public class Solution {

public static void main(String[] args) {

Solution solution = new Solution();

for (String item : solution.fireIce(1, 6)) {

System.out.print(item + " ");

}

}

private String[] fireIce(int start, int end) {

String[] res = new String[(end - start)];

int index = 0;

for (int i = start; i < end; i++) {

if (i % 3 == 0 && i % 5 == 0) {

res[index] = "Water";

} else if (i % 3 == 0) {

res[index] = "Fire";

} else if (i % 5 == 0) {

res[index] = "Ice";

} else {

res[index] = String.valueOf(i);

}

index++;

}

return res;

}

}

Q4:

a, select d.name from driver as d left join car as c on d.car = c.id where c.make = 'Honda' and c.model = 'Civic'

b, SELECT Car.make, Car.model, COUNT(Driver.id) AS drivers\_count FROM Driver JOIN Car ON Driver.car = Car.id GROUP BY Car.make, Car.model

Q5:

Take for example String buffer and String in java. String is immutable and final in Java and every modification in String creates a new String object.

A String that can be modified or changed is known as mutable String. StringBuffer are classes are used for creating mutable strings.

Summarizing that when you want to preserve data, handle multithreading, use immutable. Otherwise use mutable to save memory

Q6:

a, The drawbacks of this design would cost out of control of errorCode in the near future when there are too many errorCode added up in the program. Thus, it would hard to read code and change the errorCode.

b, Make all the errorCode in Array constant divided into specific group.

ex: private static final int[] FAN\_BELT\_ERR\_CODE = {04, 06, 07, 08, 11};

Q7:

a, This is the factory pattern design. The code can become more complicated than usual because it requires using many classes to implement this pattern.

b,

abstract class Vehicle {

abstract void drive();

}

class Truck extends Vehicle {

@Override

public void drive() {

}

}

public class GenerateVehicle {

public Vehicle getVehicle(String vehicleType) {

if (vehicleType == null) return null;

if (vehicleType.equalsIgnoreCase("Truck"))

return new Truck();

return null;

}

public static void main(String args[]) throws IOException {

GenerateVehicle vehicleFactory = new GenerateVehicle();

Vehicle truck = vehicleFactory.getVehicle("Truck");

}

}

Q9:

Array.from(document.getElementsByClassName('amount')).foreach((el) => el.addEventListener('forcus',(event) => {dosomething}))