1. (Binary String To Int) (50 Points)

Write a method called **int binaryStringToInt(String s, int start, int end),** which will transform the sub-string of s between **[start, end)** into an **int** value. The sub-string of s between [*start*, *end*) is treated as a binary number. For example, "110" is treated as binary number *110* which is *6* in decimal.

Requirements:

- You should **implement** the method **binaryStringToInt(String s, int start, int end)**.
- You should throw three types of exception in this method as below:
  - *StringIndexOutOfBoundsException*: if *start* or *end* is illegal
  - *NumberFormatException*: if the sub-string of s between [start, end) contain illegal characters
  - *ArithmeticException*: if **int** can't store **the binary number you want to transform** (Hint: the bits size of int is 32)
- You should **write a test program** to test the method. An output example is shown as below.

Example's output :

```
s:a'10101010101111111110010111001000101010bcd
start:0
end:2
result: incorrect binary number format.

s:a'10101010101111111110010111001000101010bcd
start:2
end:34
result: -1430264376

s:a'10101010101111111110010111001000101010bcd
start:2
end:38
result: out of bits size of int.

s:a'10101010101111111110010111001000101010bcd
start:5
end:18
result: 2751

s:a'10101010101111111110010111001000101010bcd
start:-1
end:18
result: string index out of bounds.

s:a'10101010101111111110010111001000101010bcd
start:5
end:100
```

result: string index out of bounds.

2. (Count for the Bill and take the meal) (50 Points)

Suppose we are having a meal with our friends at a restaurant. After choosing the food we need, we will **split the bill** for the meal. Then, we will enjoy our meal. In this work, you need to implement this situation. It contains four classes as below.

Abstract Class Food**:**

- **Fields :**
  - name : String, the name of the food
  - price : int, the price of the food
- **Methods :**
  - Food(String name, int price) : public, the constructor of Food
  - String getName() : public, return the name of the food
  - int getPrice() : public, return the price of the food
  - void setPrice(int price) : public, set the price of the food
  - void take() : public, an **abstract** method should be implemented by the subclasses. People use this method to take the food.

Class Drink **extends** Food**:**

- **Methods :**
  - Drink(String name, int price) : public, the constructor of Drink
  - void take() : public, method that **inherited** from the Base Class and **shall be implemented.**

Class Dishes **extends** Food**:**

- **Methods :**
  - Dishes(String name, int price) : public, the constructor of Dishes
  - void take() : public, method that **inherited** from the Base Class and **shall be implemented.**

Class Order:

- **Fields :**
  - foodList : List<Food>, the List of the food we ordered
  - peopleAmount : int, the amount of people who take the meal
- **Methods :**
  - Order(int peopleAmount) : public, the constructor of Order
  - void addFood(Food newFood) : public, add the food into the foodList, and print the name of the food to console
  - void setPeopleAmount(int peopleAmount) : public, set the amount of people
  - void showBill() : public, print the details of the food to console, and the splitted bill that each person should pay
  - void takeFood() : public, enjoy the food with **the sequence of adding to the foodList**

Requirements :

- Several methods in the Class **Order** may **throw an exception**, please use the **try-catch** block to deal with the possible exceptions. (Hint: You can infer them from the output of test code as below)
- You should **implement** the class and **complete** the test program as shown below.
- The Classes must contain methods and fields mentioned above, and it also works if your implement has more methods and fields than the requirement.

Test Program :

- Test Code:

```
public class OrderFoodTest {
    public static void main(String args[]){
        Order order = new Order(_____);

        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);
        order._____(_____);

        order._____(_____);
        order._____(_____);
        order._____(_____);

        for(int i = 0;i < 10;i++){
            order._____(_____);
        }
    }
}
```

- output :

```
The food "Coke" is added to the order
The food "Coffee" is added to the order
The food "Juice" is added to the order
The food "Tea" is added to the order
The food "tofu" is added to the order
The food "stir-fried vegetable" is added to the order
The food "fried chicken" is added to the order
The food "streamed fished" is added to the order
```

```
Bill of the Order:
name: Coke, price: 4 yuan
name: Coffee, price: 10 yuan
name: Juice, price: 6 yuan
name: Tea, price: 5 yuan
name: tofu, price: 15 yuan
name: stir-fried vegetable, price: 20 yuan
name: fried chicken, price: 30 yuan
name: streamed fished, price: 35 yuan
Error : Should be at least one person pay for the bill

The food "Chow mein" is added to the order

Bill of the Order:
name: Coke, price: 4 yuan
name: Coffee, price: 10 yuan
name: Juice, price: 6 yuan
name: Tea, price: 5 yuan
name: tofu, price: 15 yuan
name: stir-fried vegetable, price: 20 yuan
name: fried chicken, price: 30 yuan
name: streamed fished, price: 35 yuan
name: Chow mein, price: 35 yuan
Each Person should pay: 40 yuan

The Drink Coke is taken
The Drink Coffee is taken
The Drink Juice is taken
The Drink Tea is taken
The Dishes tofu is taken
The Dishes stir-fried vegetable is taken
The Dishes fried chicken is taken
The Dishes streamed fished is taken
The Dishes Chow mein is taken
Error : All the food have already taken
```

3. **(Animal Decorator) (Bonus question: 20Points)**

As we all know, with good training of dogs and cats, we can order them to sit down and shake hands. But it will be more interesting, when the dog and cat can conduct some scientific research. In this work, you need to use the **Decorator Pattern** to implement this situation, and formulate the six kinds of animals: Dog, Cat, TrainedDog, TrainedCat, ScientificDog, ScientificCat. It contains six classes as below.

Interface Animal**:**

- **Methods :**
  - String getType() : public, return the type of the animal
  - void setType(String type) : public, set the type of the animal
  - String getName() : public, return the name of the animal
  - void eat() : public, formulate the eating behavior, and print it to the console.
  - void bark() : public, formulate the barking behavior, and print it to the console.
  - void showSkills() : public, show the special skills it has, and print it to the console.

Class Dog **implements** Animal**:**

- **Fields :**
  - type : String, the type of the animal
  - name : String, the name of the animal
- **Methods :**
  - Dog(String name) : public, the constructor of Dog
  - String getType() : public, method that **inherited** from the Interface and **shall be implemented**
  - String getName() : public, method that **inherited** from the Interface and **shall be implemented**
  - void setType(String type) : public, method that **inherited** from the Interface and **shall be implemented**
  - void eat() : public, method that **inherited** from the Interface and **shall be implemented**
  - void bark() : public, method that **inherited** from the Interface and **shall be implemented**
  - void showSkills() : public, method that **inherited** from the Interface and **shall be implemented**

Class Cat **implements** Animal**:**

- **Fields :**
  - type : String, the type of the animal
  - name : String, the name of the animal
- **Methods :**
  - Cat(String name) : public, the constructor of Cat
  - String getType() : public, method that **inherited** from the Interface and **shall be implemented**
  - void setType(String type) : public, method that **inherited** from the Interface and **shall be implemented**
  - String getName() : public, method that **inherited** from the Interface and **shall be implemented**
  - void eat() : public, method that **inherited** from the Interface and **shall be implemented**
  - void bark() : public, method that **inherited** from the Interface and **shall be implemented**
  - void showSkills() : public, method that **inherited** from the Interface and **shall be implemented**

Class AnimalDecorator **implements** Animal**:**

- **Fields :**
  - animal : Animal, the animal that it need to decorate
- **Methods :**
  - AnimalDecorator(Animal animal) : public, the constructor of AnimalDecorator
  - String getType() : public, method that **inherited** from the Interface and **shall be implemented**
  - void setType(String type) : public, method that **inherited** from the Interface and **shall be implemented**
  - String getName() : public, method that **inherited** from the Interface and **shall be implemented**
  - void eat() : public, method that **inherited** from the Interface and **shall be implemented**
  - void bark() : public, method that **inherited** from the Interface and **shall be implemented**
  - void showSkills() : public, method that **inherited** from the Interface and **shall be implemented**

Class TrainedAnimalDecorator **extends** AnimalDecorator**:**

- **Methods :**
  - TrainedAnimalDecorator(Animal animal) : public, the constructor of TrainedAnimalDecorator
  - void shakeHands() : private, formulate the behavior of shaking hands, and print it to the console
  - void sitDown() : private, formulate the behavior of sitting down, and print it to the console
  - void showSkills() : public, method that **inherited** from the Base Class and **shall be overridden**

Class ScientificAnimalDecorator **extends** AnimalDecorator**:**

- **Methods :**
  - ScientificAnimalDecorator(Animal animal) : public, the constructor of ScientificAnimalDecorator
  - void doExperiment() : private, formulate the behavior of doing experiments, and print it to the console.
  - void writePaper() : private, formulate the behavior of writing paper, and print it to the console.
  - void showSkills() : public, method that **inherited** from the Base Class and **shall be overridden**

Requirements :

- You should **implement** the class and **complete** the test program as shown below.
- The Classes must contain methods and fields mentioned above, and it also works if your implement has more methods and fields than the requirement.

Test Program :

- Test Code:

```
public class AnimalDecoratorTest {
	public static void main(String args[]){
		Cat cat = new _____(_______);
		Dog dog = new _____(_______);
		TrainedAnimalDecorator trainedCat = new TrainedAnimalDecorator(new _____(_______));
		TrainedAnimalDecorator trainedDog = new TrainedAnimalDecorator(new _____(_______));
		ScientificAnimalDecorator scientificCat = new ScientificAnimalDecorator(new
_____(_______));
		ScientificAnimalDecorator scientificDog = new ScientificAnimalDecorator(new
_____(_______));

		________._________();
		________._________();
		________._________();
		System.out.println("");

		________._________();
		________._________();
		________._________();
		System.out.println("");

		________._________();
		________._________();
		________._________();
		System.out.println("");

		________._________();
		________._________();
		________._________();
		System.out.println("");

		________._________();
		________._________();
		________._________();
		System.out.println("");

		________._________();
		________._________();
		________._________();
		System.out.println("");
	}
}
```

- output :

```
The Cat "Tom" is eating the food
Meow Meow Meow
The Cat "Tom" don't have any skill.

The Dog "Herry" is eating the food
Woof Woof Woof
The Dog "Herry" don't have any skill.

The trained Cat "Tom" is eating the food
Meow Meow Meow
The trained Cat "Tom" is showing its skills:
    The trained Cat "Tom" is shaking hands with you.
    The trained Cat "Tom" sit down on the floor.

The trained Dog "Herry" is eating the food
Woof Woof Woof
The trained Dog "Herry" is showing its skills:
    The trained Dog "Herry" is shaking hands with you.
    The trained Dog "Herry" sit down on the floor.

The scientific Cat "Tom" is eating the food
Meow Meow Meow
The scientific Cat "Tom" is showing its skills:
    The scientific Cat "Tom" is doing experiment.
    The scientific Cat "Tom" is writing the paper.

The scientific Dog "Herry" is eating the food
Woof Woof Woof
The scientific Dog "Herry" is showing its skills:
    The scientific Dog "Herry" is doing experiment.
    The scientific Dog "Herry" is writing the paper.
```