Problem 1

Create two custom exception classes

1. NoGasException — checked;
2. ExplosionException — unchecked.

Then create a class Car with

1. fields of type int corresponding to the amount of fuel in the tank and the number of kilometers travelled. The fields are initialized with the values 35 and 0, respectively;
2. method `public void fill()`, which
   - with probability 10%, throws an exception of type ExplosionException (which we do *not* handle);
   - if there was no explosion, fills the tank with additional random number of litres of gasoline (from the range $[15, 35]$) and prints information on the current fuel level.
3. method `public void drive100km() throws NoGasException` which
   - throws an exception of type NoGasException, if the fuel level is below 10 litres;
   - if there is enough fuel, consumes 10 litres of gasoline and increments the number of kilometers travelled by 100 km;
   - prints information on the total number of kilometers travelled and the current fuel level.

Note: Messages about exceptions printed by the JVM go to the error stream. To see all messages issued by both the JVM and our program in the correct order, the program should print not to the standard output but rather to the error stream; therefore, use System.err.println instead of System.out.println.

For example, the following program

download *Excepts.java*

```
public class Excepts {
    public static void main (String[] args) {
        Car car = new Car();
        while (true) {
            try {
                car.drive100km();
            } catch(NoGasException e) {
                System.err.println(e.getMessage());
                car.fill();
            }
        }
    }
}
```

should print something like

```
100km driven, 25 liters left
200km driven, 15 liters left
300km driven, 5 liters left
Only 5 liters. Must fill the tank
After filling 38
400km driven, 28 liters left
500km driven, 18 liters left
600km driven, 8 liters left
Only 8 liters. Must fill the tank
After filling 27
700km driven, 17 liters left
800km driven, 7 liters left
Only 7 liters. Must fill the tank
After filling 29
900km driven, 19 liters left
1000km driven, 9 liters left
Only 9 liters. Must fill the tank
After filling 38
1100km driven, 28 liters left
1200km driven, 18 liters left
1300km driven, 8 liters left
Only 8 liters. Must fill the tank
After filling 24
1400km driven, 14 liters left
1500km driven, 4 liters left
Only 4 liters. Must fill the tank
After filling 26
1600km driven, 16 liters left
1700km driven, 6 liters left
Only 6 liters. Must fill the tank
Exception in thread "main" ExplosionException: BOOOOOOM
        at Car.fill(Excepts.java:20)
        at Excepts.main(Excepts.java:9)
```

Problem 2

Define four classes representing exceptions: three checked ones:

- NoSuchCustomerException,
- NegativeAmountException,
- InsufficientFundsException,

and one unchecked,

- BankruptcyException.

Then define a **Customer** class with two non-static private fields, balance and name, and two constructors: one that takes name and balance, and another that takes only name, with balance then defaulting to 100. The class has two methods:

- **deposit(int amount)**, which increases balance by the given amount but first checks if it is negative; if it is, it throws a **NegativeAmountException**;
- **withdraw(int amount)**, which decreases balance by the given amount but first checks if it is negative (throwing **NegativeAmountException** if it is) and whether the requested amount exceeds the sum of the customer's funds and the allowed overdraft MAX_DEBET; if it does, it throws an **InsufficientFundsException**.

The **Bank** class has a static public final field MAX_DEBET (e.g., 50), an array of customers passed to the constructor, and methods

- **getCustomerByName(String name)**, which returns the customer with that name from the array or throws a **NoSuchCustomerException** if there is none;
- **deposit(String name, int amount)**, which calls the **deposit** method the customer with the given name and handles all exceptions that might occur (if any exception occurs, a message is printed and the transaction is not carried out);
- **withdraw(String name, int a)**, which calls the **withdraw** method on the customer with the given name and also handles all possible exceptions.

After each withdrawal, the total sum of all funds held by customers is checked, and if it is negative, the bank goes bankrupt by throwing an unchecked and unhandled **BankruptcyException**.

Executing the following program

download *BankExceptions.java*

```
public class BankExceptions {
    public static void main(String[] args) {
        Customer[] customers = {
            new Customer("Jane", 60), new Customer("John", 20),
            new Customer("Bill"), new Customer("Sue")
        };
        Bank bank = new Bank(customers);
        bank.deposit("Carol", 20);
        bank.deposit("Bill", 10);
        bank.withdraw("Jane", 110);
        bank.withdraw("Sue", 140);
        bank.deposit("Sue", -40);
        bank.deposit("John", 10);
        bank.withdraw("Jane", 50);
        bank.withdraw("Bill", 90);
    }
}
```

should print something like

```
Cancelled: No such customer: Carol
Bill: deposit 10
Jane: withdrawal 110
Sue: withdrawal 140
Cancelled: amount negative
John: deposit 10
Cancelled: insufficient funds
Bill: withdrawal 90
Exception in thread "main" BankruptcyException:
            Bank went bankrupt!!!
        at Bank.checkAssets(BankExceptions.java:16)
        at Bank.withdraw(BankExceptions.java:52)
        at BankExceptions.main(BankExceptions.java:116)
```

Note: In order for the messages to appear in the correct order, all calls to the **print(ln)** method should be made to System.err rather than System.out.