1. (30 Points) Synchronization of Selling Tickets at the Window

Write a Java multithreaded program to simulate three ticket window at the same time to sell 20 tickets. To ensure that the same ticket will not be sold, a Java multithreaded synchronization lock is required.

Sample Output:

```
Window 1: ticket 1
Window 3: ticket 2
Window 2: ticket 3
Window 2: ticket 4
Window 3: ticket 5
Window 3: ticket 6
Window 1: ticket 7
Window 3: ticket 8
Window 2: ticket 9
Window 2: ticket 10
Window 3: ticket 11
Window 1: ticket 12
Window 1: ticket 13
Window 1: ticket 14
Window 1: ticket 15
Window 3: ticket 16
Window 3: ticket 17
Window 3: ticket 18
Window 3: ticket 19
Window 3: ticket 20
Sold Out
```

2. (30 Points) Multi-Thread: Bank System

Design a program using interface Runnable to simulate a bank system. There is an account A with 100 dollars. Two different accounts, i.e., B and C, tend to transfer $100 to account A and repeats 10 times. When doing the transferring operation, the system gets the balance of account A and then update it by adding the money that another account transfers to it.

Finally, your program is required to do a statistics and print the result after all transfers to check whether it runs properly.

Sample Output:

```
balance of account A: $100
account B transfer $100 to account A, balance of account A: $200
account B transfer $100 to account A, balance of account A: $300
account B transfer $100 to account A, balance of account A: $400
account B transfer $100 to account A, balance of account A: $500
account B transfer $100 to account A, balance of account A: $600
account B transfer $100 to account A, balance of account A: $700
account C transfer $100 to account A, balance of account A: $800
account B transfer $100 to account A, balance of account A: $900
account B transfer $100 to account A, balance of account A: $1000
account B transfer $100 to account A, balance of account A: $1100
account B transfer $100 to account A, balance of account A: $1200
account C transfer $100 to account A, balance of account A: $1300
account C transfer $100 to account A, balance of account A: $1400
account C transfer $100 to account A, balance of account A: $1500
account C transfer $100 to account A, balance of account A: $1600
account C transfer $100 to account A, balance of account A: $1700
account C transfer $100 to account A, balance of account A: $1800
account C transfer $100 to account A, balance of account A: $1900
account C transfer $100 to account A, balance of account A: $2000
account C transfer $100 to account A, balance of account A: $2100
account B transfer $1000 to account A in total
account C transfer $1000 to account A in total
balance of account A: $2100
```

3. (40 Points) Inter-Thread Synchronization

(1) Write a class called Producer which implements Runnable class and contains:

- Method run that produces content. The Producer needs to use run method to produce 10 content, “12345” for odd iteration, “abcde” for even. And only the Customer consumes the last content, the producer can produce a new one.

(2) Write a class called Customer which implements Runnable class and contains:

- Method run that consumes content. The producer needs to use run method to consume 10 contents, and the Customer can only consumes the last produced content.

(3) Write a class called Master.

- Method master to call Producer and Customer.

The code of class Info, Producer and Customer are as follows:

```
public class Producer implements Runnable{
        @Override
        public void run() {
                .....
        }
}
public class Customer implements Runnable{
        @Override
        public void run() {
              .......
      }
}
public class Master{
        public void master() {
                .......
      }
}
```

Sample Output:

```
Set 1-> Item1 : 12345
Get 1-> Item1 : 12345
Set 2-> Item2 : abcde
Get 2-> Item2 : abcde
Set 3-> Item1 : 12345
Get 3-> Item1 : 12345
Set 4-> Item2 : abcde
Get 4-> Item2 : abcde
Set 5-> Item1 : 12345
Get 5-> Item1 : 12345
Set 6-> Item2 : abcde
Get 6-> Item2 : abcde
Set 7-> Item1 : 12345
Get 7-> Item1 : 12345
Set 8-> Item2 : abcde
Get 8-> Item2 : abcde
Set 9-> Item1 : 12345
Get 9-> Item1 : 12345
Set 10-> Item2 : abcde
Get 10-> Item2 : abcde
```

4. (Bonus Question: 20 Points) Deadlock: Dining Philosopher Problem

Five silent philosophers sit at a round table with bowls of spaghetti shown in Fig. 1. Forks are placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks.

Now you are required to do the following tasks:

1) Write a program that could cause a deadlock.

2) Write another program that can avoid the deadlock so that all the philosophers can eat alternately.

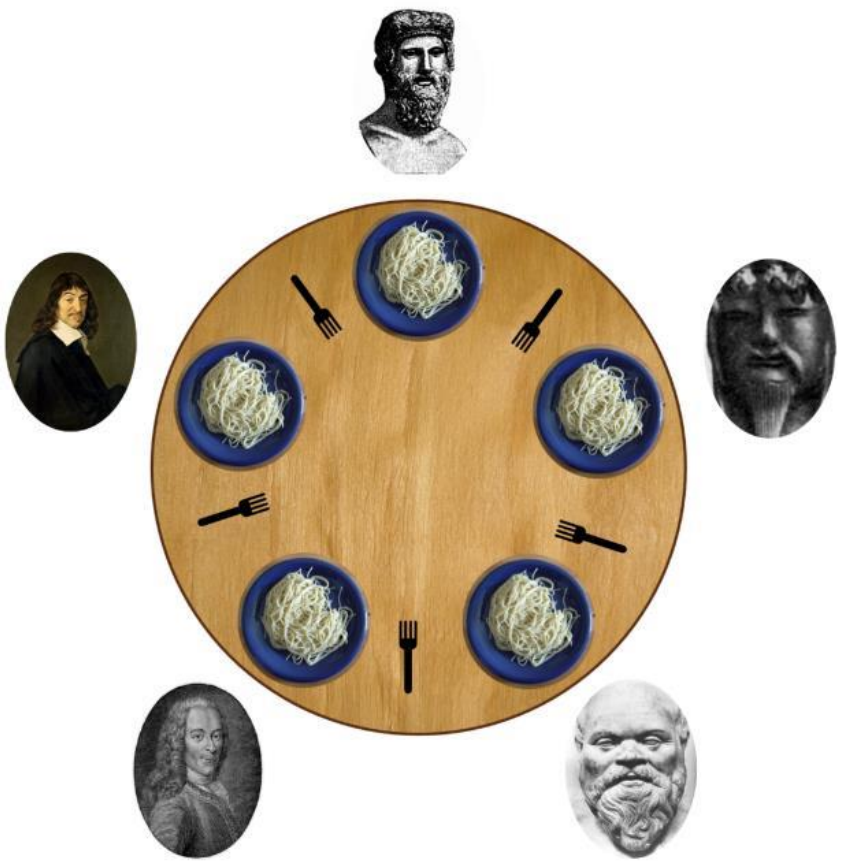
Fig. 1: The Dining Philosophers Problem.