1. Name and describe 5 states of a process.

New state: The process is being created

Ready state: Instructions are being executed

Waiting state: The process is waiting for some event to occur

Ready state: The process is waiting to be assigned to a processor

Terminated state: The process has finished execution

1. Describe four benefits of multithreaded programming.
2. Responsiveness. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation.
3. Resource sharing. Threads share the memory and the resources of the process to which they belong by default.
4. Economy. Because threads share the resources of the process to which they belong, it is more economical to create and manage processes than threads.
5. Scalability. A single-threaded process can run on only one processor, regardless how many are available.
6. Multi thread synchronization practice, Bank System:

a. Write a BankAccount class which implements:

i. deposit(int amount)

ii. withdraw(int amount)

iii. getBalance()

b. Write a main function to simulate the bank system. Use multi threading to randomly deposit or withdraw random amount.

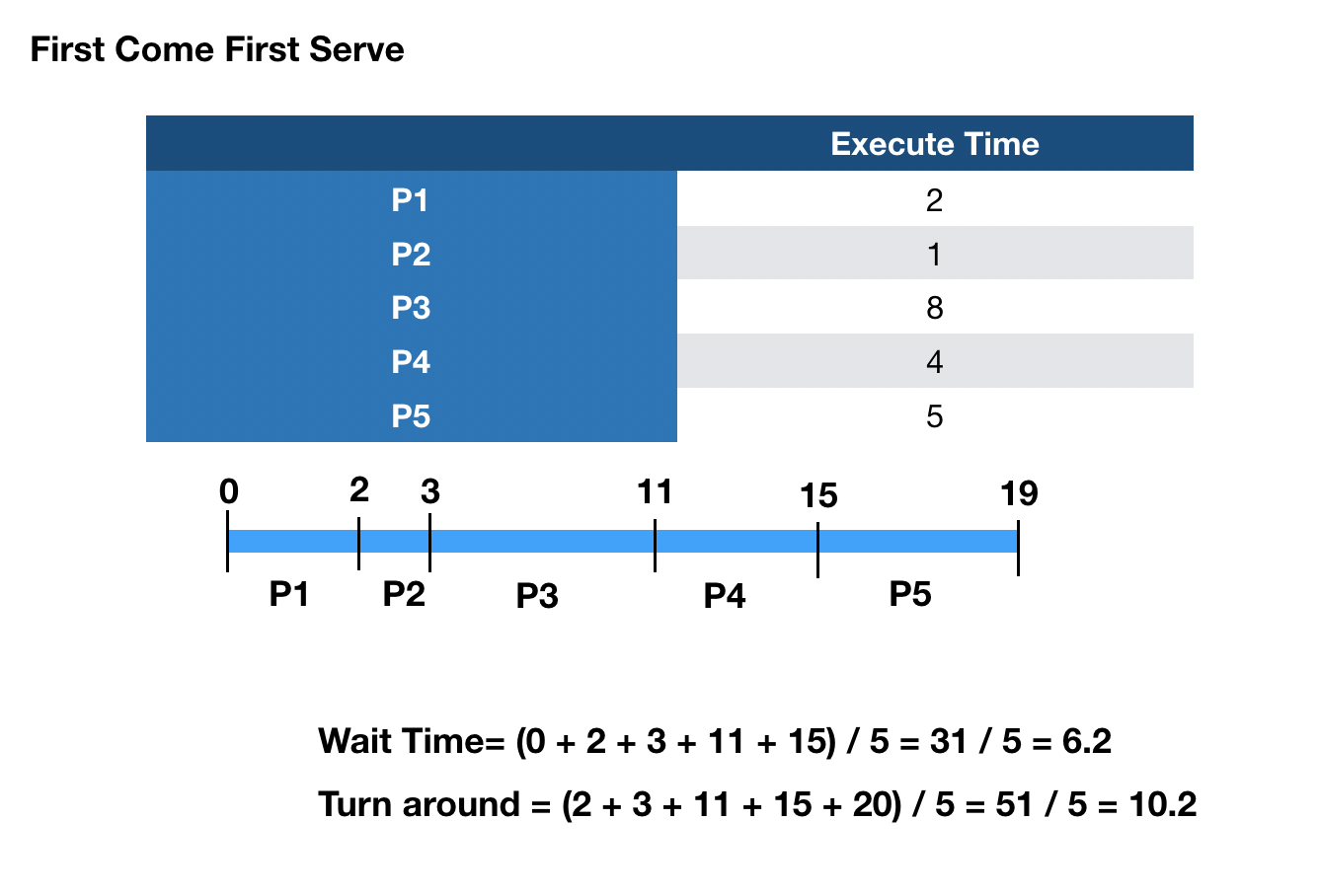
c. Handle the synchronization and guarantee the balance is always >= 0.

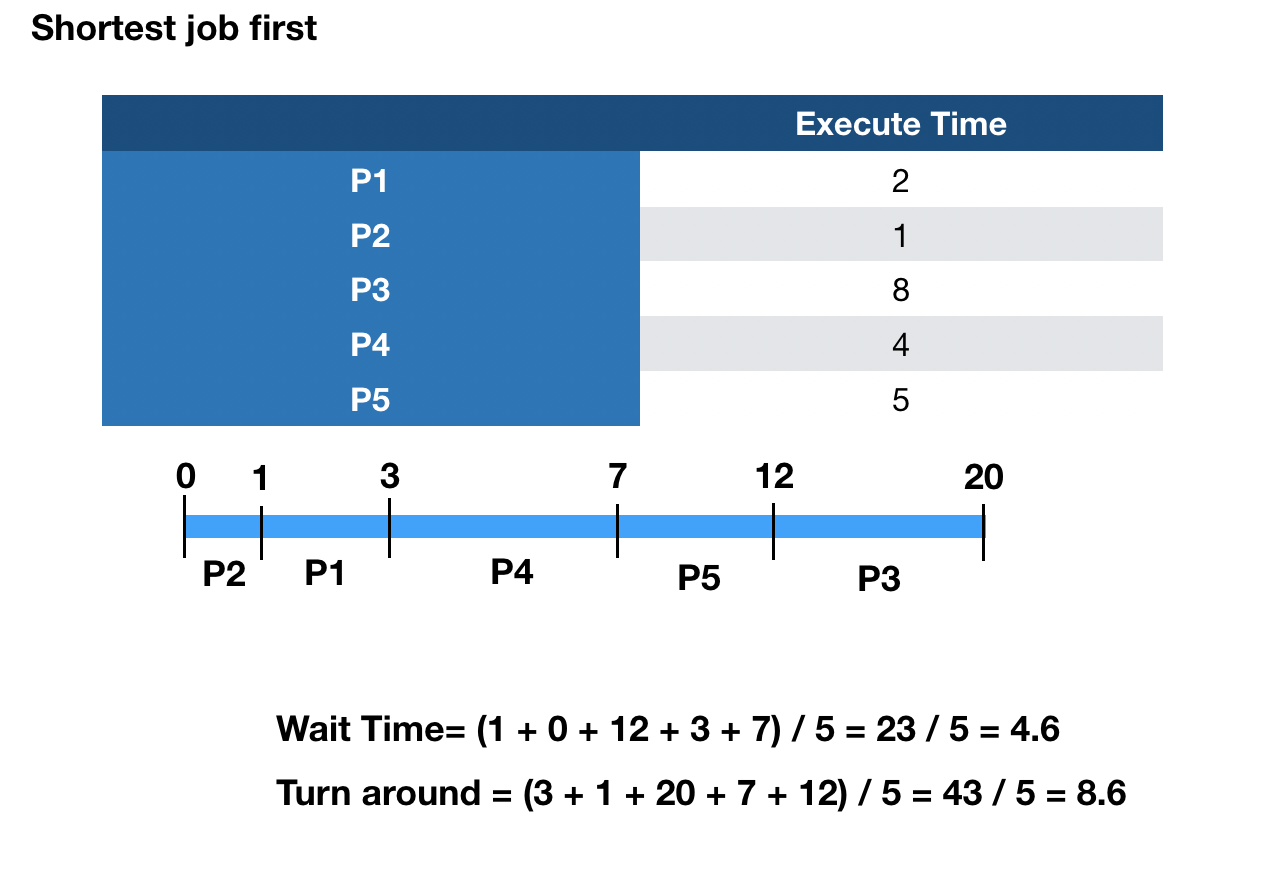
import java.util.Random;  
  
public class BankAccount {  
  
 private int balance;  
 private final Object lock = new Object();  
 static Random *random* = new Random();  
  
 public static void main(String[] args){  
  
 BankAccount bankAccount = new BankAccount();  
 for (int i = 0; i < 5; i++) {  
 new DepositThread(bankAccount).start();  
 }  
 for (int i = 0; i < 5; i++) {  
 new WithdrawThread(bankAccount).start();  
 }  
 }  
  
 public BankAccount(){  
 this.balance = 0;  
 }  
  
 public BankAccount(int amount){  
 this.balance = amount;  
 }  
 public void deposit(int amount) {  
 balance += amount;  
 }  
 public void withdraw(int amount) {  
 synchronized (lock) {  
 if (balance < amount) System.*out*.println("Balance is not enough");  
 else balance -= amount;  
 }  
 }  
 public int getBalance() {  
 return balance;  
 }  
 static class DepositThread extends Thread {  
 private BankAccount bankAccount;  
 public DepositThread(BankAccount bankAccount){  
 this.bankAccount = bankAccount;  
 }  
 @Override  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 bankAccount.deposit(*random*.nextInt());  
 System.*out*.println("Deposit: " + bankAccount.getBalance());  
 }  
 }  
 }  
 static class WithdrawThread extends Thread {  
 private BankAccount bankAccount;  
 public WithdrawThread(BankAccount bankAccount){  
 this.bankAccount = bankAccount;  
 }  
 @Override  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 bankAccount.deposit(*random*.nextInt());  
 System.*out*.println("WithDraw: " + bankAccount.getBalance());  
 }  
 }  
 }  
}

1. Consider the following set of processes, with the length of the CPU burst given in milliseconds:

P1 2 sec, P2 1sec, P3 8sec, P4 4 sec, P5 5sec.

Find the turn-around time, and waiting time for FCFS and SJF algorithms.





1. Bankers algorithm:

                Allocation                Max                               Available

             A    B    C    D          A    B    C    D              A    B    C    D

P0       0     0     1    2           0    0     1    2              1    5     2    0

P1       1     0     0    0           1    7     5    0

P2       1     3     5    4           2    3     5    6

P3       0     6     3     2          0    6     5    2

P4       0     0      1    4          0    6     5    6

Is there a safe sequence in which the processes can execute without deadlock?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** |
| **P0** | 0 | 0 | 0 | 0 |
| **P1** | 0 | 7 | 5 | 0 |
| **P2** | 1 | 0 | 0 | 2 |
| **P3** | 0 | 0 | 2 | 0 |
| **P4** | 0 | 6 | 4 | 2 |

P0 -> ( 1 5 3 2 ) -> P2 -> ( 2 8 8 6 ) -> P1-> ( 3 8 8 6 ) -> P3 -> ( 3 14 11 8 ) -> P4