JAVA THREADING ASSIGNMENT

3rd June 2025

Abhik Chakraborty (Abhik.Chakraborty@bounteous.com)

1. **Print Odd and Even Numbers Using Two Threads**
   You are tasked with writing a Java program that prints odd and even numbers in sequence using two separate threads. One thread should print odd numbers, and the other thread should print even numbers. The threads should coordinate to ensure the numbers are printed in the correct order.
   a. Requirements
      i. Two Threads: One thread prints odd numbers, and the other thread prints numbers.
      ii. Synchronization: The threads must coordinate to print the numbers in sequence without any overlap or missing numbers.
      iii. Range: The program should print numbers from 1 to a specified maximum value.
      Implementation
         · **OddEvenPrinter Class:** Manages the printing of odd and even numbers using two threads.
         · **OddThread Class:** Represents the thread that prints odd numbers.
         · **EvenThread Class:** Represents the thread that prints even numbers.

Solution:
Print Odd and Even Numbers Using Two Threads
Print numbers in order from 1 to N using two threads – one for odd, one for even.
Parent Class -> OddEvenPrinter
Child Classes which extends form Thread Class
->OddThread
->EvenThread

OddEvenPrinter.java

```java
package assignment;

public class OddEvenPrinter {  no usages
    int max;  3 usages
    int number = 1;  6 usages

    OddEvenPrinter(int max){  no usages
        this.max = max;
    }
    public synchronized void printOdd() {  no usages
        while (number <= max) {
            if (number % 2 == 1) {
                System.out.println("Odd: " + number++);
                notify();
            } else {
                waitForTurn();
            }
        }
        notify();
    }
```

OddThread.java

```java
package assignment;

public class OddThread extends Thread{  no usag
    OddEvenPrinter printer;  2 usages

    public OddThread(OddEvenPrinter printer){
        this.printer = printer;
    }
    public void run() {
        printer.printOdd();
    }
}
```

EvenThread.java

```java
package assignment;

public class EvenThread extends Thread{  no usage
    OddEvenPrinter printer;  2 usages

    EvenThread(OddEvenPrinter printer){  no usage
        this.printer = printer;
    }

    public void run() {  no usages
        printer.printEven();
    }
}
```

Main Class

```java
package assignment;

public class Main {
    public static void main(String[] args) {
        OddEvenPrinter printer = new OddEvenPrinter( max: 10);
        new OddThread(printer).start();
        new EvenThread(printer).start();
    }
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-21\bin\jav
Odd: 1
Even: 2
Odd: 3
Even: 4
Odd: 5
Even: 6
Odd: 7
Even: 8
Odd: 9
Even: 10

Process finished with exit code 0
```

2. **Bridge Crossing with Shared Token**

There are two cities, City A and City B, connected by a bridge. Only one person can cross the bridge at a time. To cross the bridge, a person must take a token from one end and deposit it at the other end. There is only one token available, and it must be shared by all residents of both cities. Initially, the token is in City B. Residents of City B must use the token to travel to City A first, and only then can residents of City A use the token to travel to City B.

Requirements

1. Bridge: A shared resource that only one person can use at a time.

2. Token: A token with one permit representing the token that controls access to the bridge.

3. Direction Control: A mechanism to ensure that the token is used by residents of City B to travel to City A first, and then by residents of City A to travel to City B.

4. Implement Waiting Queue: Implement queue for both cities so that people will get the turn to cross the city.

Implementation

1. Bridge Class: Manages the token and the direction control.

2. Person Class: Represents a person who wants to cross the bridge.

3. BridgeManagement Class: Creates instances of Person and starts their threads.

Solution:

Bridge Crossing with Shared Token

People from two cities share a bridge with a token. Only one crosses at a time.

Bridge ->parent class

Person -> extends from Thread

```java
package assignment.BrideCrossing;

public class Bridge {  no usages
    private boolean isTokenInCityB = true;  5 usages
    public synchronized void cross(String city, String personName) {  no usages
        try {
            while ((city.equals("CityA") && isTokenInCityB) || (city.equals("CityB") && !isTokenInCityB)) {
                wait();
            }
            System.out.println(personName + " from " + city + " is crossing the bridge.");
            Thread.sleep( millis: 1000);

            isTokenInCityB = !isTokenInCityB;
            System.out.println(personName + " reached opposite city. Token moved to " + (isTokenInCityB ? "CityB" : "CityA"));
            notifyAll();
        }
        catch (InterruptedException e){
            Thread.currentThread().interrupt();
        }
    }
}
```

Person.java

```java
package assignment.BrideCrossing;

public class Person extends Thread{  no usages
    String city;  2 usages
    Bridge bridge;  2 usages

    Person(String name, String city, Bridge bridge){  no usages
        super(name);
        this.city = city;
        this.bridge = bridge;
    }

    public void run(){
        bridge.cross(city, getName());
    }
}
```

BridgeManagement.java

Bridge.java | Person.java | BridgeManagement.java

```java
package assignment.BrideCrossing;

public class BridgeManagement {
    public static void main(String[] args) {
        Bridge bridge = new Bridge();

        for (int i = 1; i <= 5; i++) {
            new Person( name: "PersonB" + i,  city: "CityB", bridge).start();
        }

        for (int i = 1; i <= 5; i++) {
            new Person( name: "PersonA" + i,  city: "CityA", bridge).start();
        }
    }
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Users
PersonB1 from CityB is crossing the bridge.
PersonB1 reached opposite city. Token moved to CityA
PersonA5 from CityA is crossing the bridge.
PersonA5 reached opposite city. Token moved to CityB
PersonB5 from CityB is crossing the bridge.
PersonB5 reached opposite city. Token moved to CityA
PersonA1 from CityA is crossing the bridge.
PersonA1 reached opposite city. Token moved to CityB
PersonB2 from CityB is crossing the bridge.
PersonB2 reached opposite city. Token moved to CityA
PersonA4 from CityA is crossing the bridge.
PersonA4 reached opposite city. Token moved to CityB
PersonB4 from CityB is crossing the bridge.
PersonB4 reached opposite city. Token moved to CityA
PersonA2 from CityA is crossing the bridge.
PersonA2 reached opposite city. Token moved to CityB
PersonB3 from CityB is crossing the bridge.
PersonB3 reached opposite city. Token moved to CityA
PersonA3 from CityA is crossing the bridge.
PersonA3 reached opposite city. Token moved to CityB

Process finished with exit code 0
```