Birla Institute of Technology & Science – Pilani

First Semester 2015-16

Date: 22nd November, 2015 (Sunday)

ONLINE TEST (OPEN BOOK)

Course No and Name: Object Oriented Programming (CS F213)
Max. Marks: 96 Weightage:32% Time: 9:00 AM – 11:30 AM

Important Instructions

- You are not allowed to modify the code of the existing classes. You cannot add any method or any other variable in the existing classes.
- You are given a .java file named 'OnlineTest.Java'. Download it from the link given below and Save it in a local folder. Rename the file as P<IDNO>.java. For example if your ID NO is '2014A7005P' then the name of the renamed file should be 'P2014A7005.java'.
- The safekeeping of the written code is your responsibility. No query about the 'lost or misplaced' files will be considered at the end of the exam. Keep Saving your work regularly.

Car Assembly Line Simulator

The given problem deals with the simulation of a 'Car Assembly Line'. In the given problem, an 'Assembly Line' is a collection of 10 'Slots' and in each 'Slot' one car can be manufactured. The simulation program given to you contain the classes as shown in the table below.

Table: Classes Used to Simulate a 'Car Assembly Line'

S.No	Name of Class	Statement of Purpose
1	Car	Simulates the features of a Car
2	Slot	A slot holds a car. A slot has a label such as 'Slot I', 'Slot II' etc and three buttons labeled 'E','T' and 'L'. The button 'E' represents engine, 'T' represents tyres and 'L' represents light system.
3	AssemblyLine	An AssemblyLine class holds 10 fixed slots where one car is manufactured.
4	AdderThread	Adder Thread adds a car into the selected slot of an AssemblyLine class
5	EngineThread	Engine Thread fits an Engine into a car
6	TyreThread	Tyre Thread fits an Tyres into a car
7	LightThread	Light Thread fits an Lighting system into a car
8	RemoveThread	Remove Thread removes a car from a slot so that Adder Thread can add
9	OnlineTest	Driver Class

The code for the classes namely '**Car**', '**Slot**' and '**AssemblyLine**' is given to you and **you are not allowed to add or modify anything(any other instance fields or any other method) in the code of these classes**. A skeleton code (comprising the instance fields and constructor method) of the five thread classed named '**AdderThread**', '**EngineThread**', '**TyreThread**', '**LightThread**' and '**RemoveThread**' is also provided to you. **You have to complete the 'run' method of these five thread classes without adding or modifying any other instance fields or any other method in these classes.**

The explanation about the attributes and methods of three classes namely '**Car**', '**Slot**' and '**AssemblyLine**' is given on the next page.

1. Car

As mentioned earlier also, this class represents a real world 'car'. The commented description of each instance field and method is given below

```
class Car
{
        private boolean engineFitted;           // boolean variable to indicate whether engine is fitted into car or not
        private boolean tyreFitted;             // boolean variable to indicate whether tyres are fitted into car or not
        private boolean lightSystemFitted;      // boolean variable to indicate whether light system is fitted into car or not

        // Constructor Method
        Car()
        {
                engineFitted = tyreFitted = lightSystemFitted = false;      // Initially no part part is fitted into car
        }

        // Accessor Methods
        public boolean isEngineFitted()         { return engineFitted;}
        public boolean isTyreFitted()           { return tyreFitted;}
        public boolean isLightSystemFitted()    { return lightSystemFitted;}
        public boolean iCarReady()              { return engineFitted && tyreFitted && lightSystemFitted;  }

        // Methods to fit Engine, Tyres and Light System into Car

        public void fitEngine()                 { engineFitted = true;}
        public void fitTyres()                  { tyreFitted = true;  }
        public void fitLightSystem()            { lightSystemFitted = true;}

} // End of Class Car
```

2. Slot

This class represents a slot where one car is manufactured. A slot is simulated using four attributes namely 'slotHeadingLabel', 'engineButton', 'tyreButton', 'lightSystemButton' and 'car'. The description of each field is given in the following commented code.

```
class Slot
{
        private JLabel slotHeadingLabel;        // Slot Heading Label (For Example "Slot I","Slot II" etc. )
        private JButton engineButton;           // Engine Button of Slot
        private JButton tyreButton;             // Tyre Button of Slot
        private JButton lightSystemButton;      // Light System Button of Slot
        private Car     car;                    // Car held by the Slot

        // Constructor Method
        Slot(JLabel slotHeadingLabel, JButton engineButton, JButton tyreButton, JButton lightSystemButton)
        {
                        this.slotHeadingLabel   = slotHeadingLabel;
                        this.engineButton       = engineButton;
                        this.tyreButton         = tyreButton;
                        this.lightSystemButton  = lightSystemButton;
                        this.car                = null;         // <-- Initially Slot Holds No Car
        }
```

```
        // Accessor Methods
        public JLabel getSlotHeadingLabel()         { return this.slotHeadingLabel;}
        public JButton getEngineButton()            { return this.engineButton;}
        public JButton getTyreButton()              { return this.tyreButton;}
        public JButton getLightSystemButton()       { return this.lightSystemButton;}
        public Car getCar()                         { return this.car; }

        // Method to Add a car into a Slot
        public void addCarToSlot(Car c)
        {
                this.car = c;
        }

        // Method to Remove a car From a Slot
        public void removeCarFromSlot()
        {
                this.car = null;
        }

        // Method to check if a slot holds a car or empty
        public boolean isEmpty()
        {
                        if (this.car == null )      return true;
                        else                        return false;
        }
}// End of Class Slot
```

3. AssemblyLine

An AssemblyLine class has exactly 10 slots and methods to initialize and start it. The description of each method is given below

```
class AssemblyLine
{
        private Slot[] slots;                       // Slots of an Assembly Line
        private boolean assemblyLineInitialized;    // boolean variable to check if an an Assembly Line is Initialized or not
        private boolean assemblyLineStarted;        // boolean variable to check if an an Assembly Line is Started or not

        // Constructor Method
        AssemblyLine()
        {
                slots = new Slot[10];               // An assembly Line has 10 fixed Slots
                assemblyLineInitialized=assemblyLineStarted=false; // Initial Staus : Not Initialized and Not Started
        }

        // Accessor Methods
        public synchronized Slot[]  getSlots()              { return slots;}
        public synchronized boolean isInitialized()         { return assemblyLineInitialized;}
        public synchronized boolean isStarted()             { return assemblyLineStarted;}

        // Methods to Initialize and Start an Assembly Line
        public void initialize()            {       assemblyLineInitialized = true;     }
        public void start()                 {       assemblyLineStarted = true;         }

}// End of class AssemblyLine
```

When the simulation application is run, it displays a frame window as shown in Figure 1 on the next page. The whole window is divided into three main panels named '**Car Assembly Control Panel**', '**Slot Panel**' and '**Thread Panel**' shown in red colors. A 'Car Assembly Control Panel' has a label ('Car Assembly Controller) and two buttons ('Initialize' and 'Start'). A 'Slot Panel' has ten separate slots and each slot has four components as 'slotHeadingLabel' (shown as 'Slot I', 'Slot II' etc.), 'engineButton' (labeled as 'E' in all slots in Figure 1), 'tyreButton' (labeled as 'T' in all slots in Figure 1) and 'lightSystemButton' (labeled as 'L' in all slots of Figure 1). A 'Thread Panel' has five buttons representing five threds and these re labeled as 'Adder Thread', 'Engine Thread', 'Tyre Thread', 'Light Thread' and 'Remove Thread'. A 'Thread Panel' also has text area field to show the trace of each thread execution.

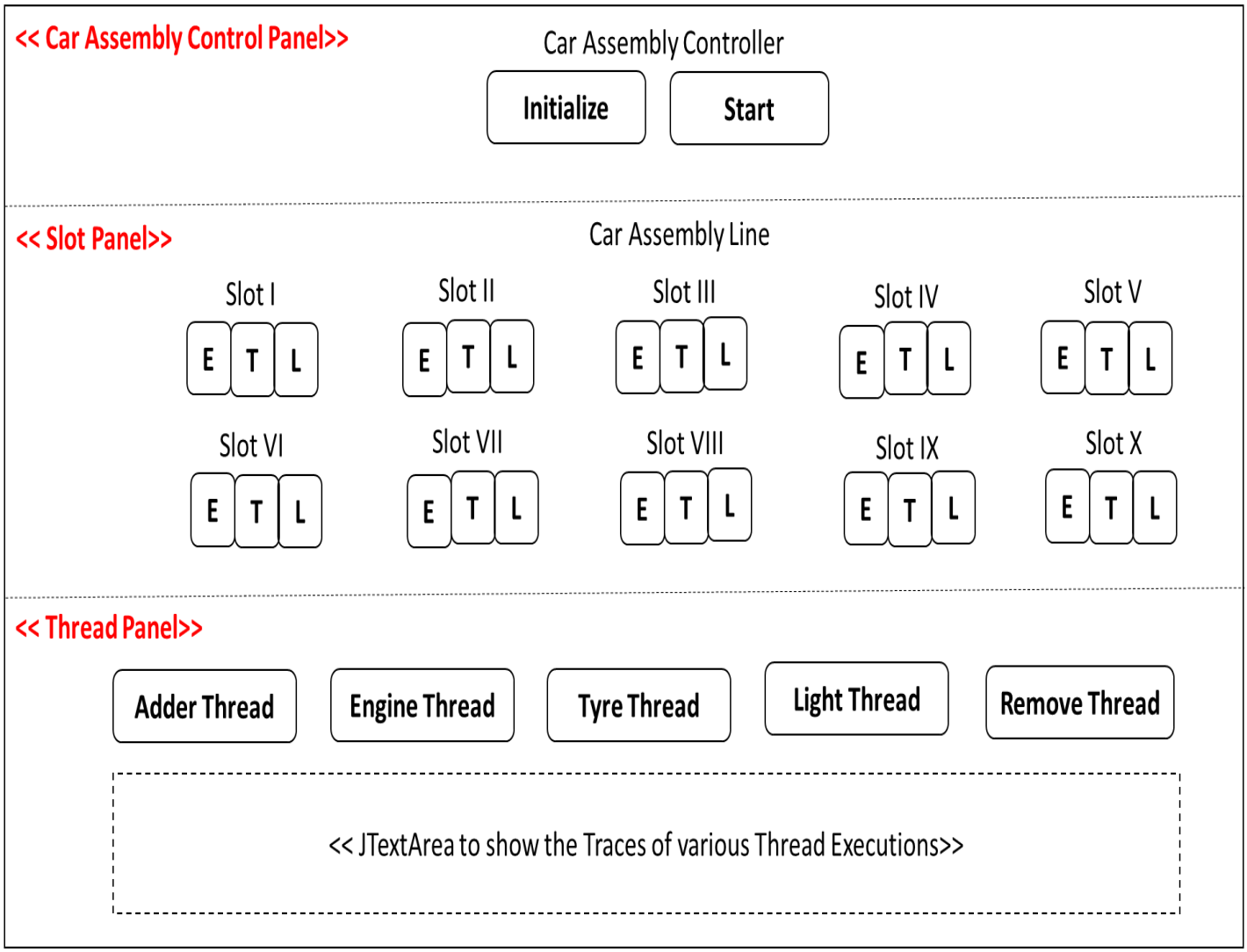

Figure 1. Car Assembly Line Frame Window

During execution when the frame window is initially displayed, only 'Initialize' Button is active whereas all other buttons are shown as inactive (You can check via execution).

Tasks You Have To Do

Task 1: **Complete the run() method of AdderThread class**

The traces of the execution of the Adder Thread under two situations (i) when assembly line is not started and (ii) when assembly line is started is shown below

Table : Execution Trace of Adder Thread

AdderThread Execution Trace shown in TextArea Field When assembly line is not started	AdderThread Execution Trace Trace shown in TextArea Field When assembly line is started
ADDER THREAD RUNNING Iteration Number:1 Adder Thread Going in Waiting State Iteration Execution Time:2008 Milli Seconds	ADDER THREAD RUNNING Iteration Number:2 Adder Thread Checking For Slot No: 1 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 2 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 3 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 4 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 5 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 6 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 7 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 8 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 9 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Checking For Slot No: 10 Status: Slot Empty .. Adding Car Status: Car Added Adder Thread Going in Waiting State Iteration Execution Time:10029 Milli Seconds

The job of the adder thread is to add a car into various slots of an assembly line only when assembly line is started otherwise it goes into waiting state. During each execution it also indicates its iteration number (how many times in the past the thread has been allotted CPU) and before going to waiting state it displays the time (in milliseconds) that it spends during the execution of the iteration in the text area. During each iteration, the minimum time for which an adder thread should execute is 2 seconds (2000 ms). During period of execution, the color of the respective button should remain green and before going to waiting state the color should be set to gray.

During each iteration when assembly line is started, an adder thread checks each slot in sequence and records the execution trace in text area as follows
If slot already has a car then it displays the message as per following format

Adder Thread Checking For Slot No: <slot-no>
Status: <mmm>

Where <mmm> is "Status: Slot Already Full"

If the slot does not have a car then it displays the message in text area as per following format

Adder Thread Checking For Slot No: <slot-no>
Status: <mmm-1>
Status:<mmm-2>

Where <mmm-1> is "Status: Slot Empty .. Adding Car
<mmm-2> is "Status: Car Added"
There is a gap of 2000 milliseconds in the display of <<mmm-1>> and <<mmm-2>>.
At the end of iteration, the execution time is displayed in the text area.

Task 2: Complete the run() method of EngineThread class

The job of the engine thread is to add an engine into car into various slots of an assembly line only when assembly line is started and otherwise it will go to wait state
Minimum execution time for this thread is also 2000 ms.
During each iteration when assembly line is started, an engine thread checks each slot in sequence and records the execution trace in text area as follows

1. If the selected slot does not have a car

   "Engine Thread Checking For Slot No: "
   Status: Slot Empty
2. If the selected slot have a car but has already engine fitted

   "Engine Thread Checking For Slot No: "
   Status: Engine Already Fitted
3. If the selected slot have a car but has no engine fitted

   "Engine Thread Checking For Slot No: "
   Status: ...Adding Engine into Car → Line X
   Status: Engine Added To Car → Line Y

   Line Y is displayed after a gap of 2000 ms . Note that engine thread takes 2000ms time to fit engine into car

Before going into waiting state, the following message is recorded in the text area.

'Engine Thread Going in Waiting State'
Iteration Execution Time: <x>

Task 3: Complete the run() method of TyreThread class

The job of the tyre thread is to add a tyres into car into various slots of an assembly line only when assembly line is started and otherwise it will go to wait state.
Minimum execution time for this thread is also 2000 ms.
During each iteration when assembly line is started, a tyre thread checks each slot in sequence and records the execution trace in text area as follows

1. If the selected slot does not have a car
   "Tyre Thread Checking For Slot No: "
   Status: Slot Empty
2. If the selected slot have a car but does not have engine fitted
   "Tyre Thread Checking For Slot No: "
   Status: Engine Not Fitted
3. If the selected slot have a car and engine but has tyres already fitted
   "Tyre Thread Checking For Slot No: "
   Status: Tyres Already Fitted

4. If the selected slot have a car and engine but has tyres already fitted
   "Tyre Thread Checking For Slot No: "
   Status: ...Adding Tyres into Car → Line X
   Status: Tyres Added To Car → Line Y

   Line Y is displayed after a gap of 2000 ms . Note that tyre thread takes 2000ms time to fit tyres into car

   Before going into waiting state, the following message is recorded in the text area.
   'Tyre Thread Going in Waiting State'
   Iteration Execution Time: <x>

Task 4: Complete the run() method of LightThread class

The job of the light thread is to add a light system into car into various slots of an assembly line only when assembly line is started and otherwise it will go to wait state.
Minimum execution time for this thread is also 2000 ms.
During each iteration when assembly line is started, a light thread checks each slot in sequence and records the execution trace in text area as follows

1. If the selected slot does not have a car
   "Light System Thread Checking For Slot No: "
   Status: Slot Empty
2. If the selected slot have a car but does not have engine fitted
   "Light System Thread Checking For Slot No: "
   Status: Engine Not Fitted
3. If the selected slot have a car and engine but no tyres fitted
   "Light System Thread Checking For Slot No: "
   Status: Tyres Not Fitted
4. If the selected slot have a car, engine and tyres fitted but has light system already fitted
   "Light System Thread Checking For Slot No: "
   Status: Light System Already Fitted
5. If the selected slot have a car, engine and tyres fitted but has no light system fitted
   "Light System Thread Checking For Slot No: "
   Status: ... Adding Light System into Car → Line X
   Status: Light System Added into Car → Line Y

   Line Y is displayed after a gap of 2000 ms . Note that light system thread takes 2000ms time to fit light system into car

   Before going into waiting state, the following message is recorded in the text area.
   'Light System Thread Going in Waiting State'
   Iteration Execution Time: <x>

Task 5: Complete the run() method of RemoveThread class

The job of the remove thread is to remove a car from various slots of an assembly line only when assembly line is started and otherwise it will go to wait state.
Minimum execution time for this thread is also 2000 ms.
During each iteration when assembly line is started, a remove thread checks each slot in sequence and records the execution trace in text area as follows

1. If the selected slot does not have a car
   "Remove Thread Checking For Slot No: "
   Status: Slot Empty
2. If the selected slot have a car but does not have engine fitted
   "Remove Thread Checking For Slot No: "
   Status: Engine Not Fitted
3. If the selected slot have a car and engine but no tyres fitted
   "Remove Thread Checking For Slot No: "
   Status: Tyres Not Fitted
4. If the selected slot have a car, engine and tyres fitted but has no light system fitted
   "Remove Thread Checking For Slot No: "
   Status: Light System not Fitted
5. If the selected slot have a car, engine, tyres and light system fitted
   "Remove Thread Checking For Slot No: "
   Status: ... Removing Car From The Slot → Line X
   Status: Car Removed From The Slot → Line Y
   Line Y is displayed after a gap of 2000 ms . Note that light system thread takes 2000ms time to fit light system into car

   Before going into waiting state, the following message is recorded in the text area.
   'Light System Thread Going in Waiting State'
   Iteration Execution Time: <x>

Note:

- Each thread sets its color to green during execution of run and before going to waiting state it sets its color gray.
- Adder Thread changes the slot's all the three 'E' (representing Engine),'T' (representing tyres) and 'L' (representing lights) buttons to RED color only if the slot is empty.
- Engine thread changes the color of only 'E' button to GREEN if engine is added
- Tyre Thread changes the color of only 'T' button to GREEN if tyres are added
- LightThread changes the color of only 'L' button to GREEN if light system is added to car
- RemoveThread changes the slot's all the three 'E' (representing Engine),'T' (representing tyres) and 'L' (representing lights) buttons to GRAY color if car is ready
- Minimum execution time for each thread is 2000 ms.

Some Suggestions:

1. System.currentTimeMillis() method returns the system time in milli seconds

2. x.wait(); forces the thread to wait upon object pointed to by ‘x’. Similarly, x.notify() will notify all threads who waiting to acquire lock upon object pointed to by ‘x’.