Java Threads

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. This hands-on lab takes you through the basics of using Java threading.

Resources

- **Threads section** of Java tutorial
- **Thread dump analyzer 2.0** from java.net

Exercises

Exercise 1: Extending Thread class

In this exercise, you are going to learn how to create and start a thread execution by writing a class that extends Thread class. You will learn how to start the thread by either not having the start() method in the constructor of the subclass or having it in the constructor of the subclass.

1. **The start() method is NOT in the constructor of the subclass**
2. **The start() method is in the constructor of the subclass**

(1.1) The start() method is NOT in the constructor of the subclass

0. Start NetBeans IDE if you have not done so yet.
1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ExtendThreadClassTest0** as project name.
- For **Create Main Class** field, type in **ExtendThreadClassTest0**. (Figure-1.10 below)
- Click **Finish**.

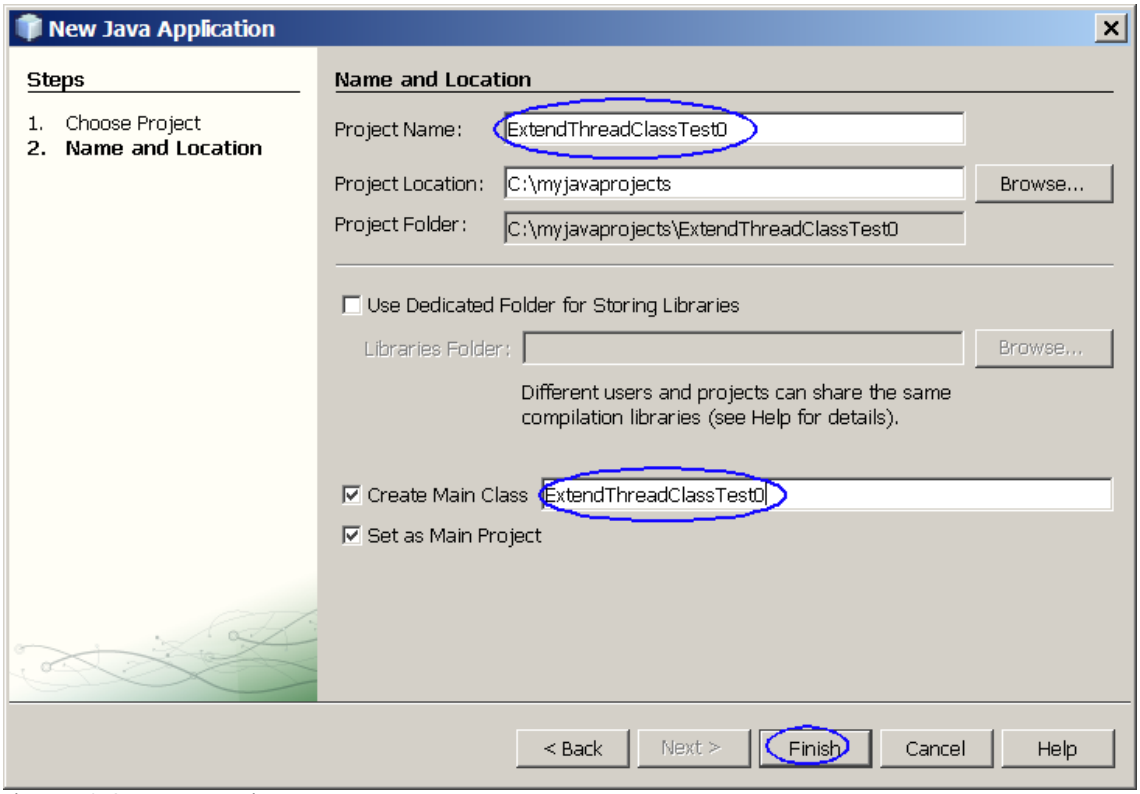


Figure-1.10: Create a new project

- Observe that **ExtendThreadClassTest0** project appears and IDE generated **ExtendThreadClassTest0.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ExtendThreadClassTest0.java** as shown in Code-1.11 below. Study the code by paying special attention to the bold fonted parts. Note that the **start()** is invoked after the object instance of **PrintNameThread** class is created.

```
public class ExtendThreadClassTest0 {

   public static void main(String args[]) {

      // Create object instance of a class that is subclass of Thread class
      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt1 =
            new PrintNameThread("A");
```

```
        // Start the thread by invoking start() method
        System.out.println("Calling start() method of " + pnt1.getName() + " thread");
        pnt1.start();

    }
}
```

Code-1.11: ExtendThreadClassTest0.java

3. Write **PrintNameThread.java** as shown in Code-1.12 below.

```
// Subclass extends Thread class
public class PrintNameThread extends Thread {

    PrintNameThread(String name) {
        super(name);
    }

    // Override the run() method of the Thread class.
    // This method gets executed when start() method
    // is invoked.
    public void run() {
        System.out.println("run() method of the " + this.getName() + " thread is called" );

        for (int i = 0; i < 10; i++) {
            System.out.print(this.getName());
        }
    }
}
```

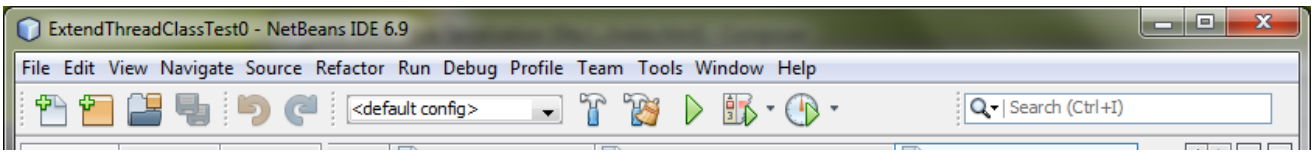
Code-1.12: PrintNameThread.java

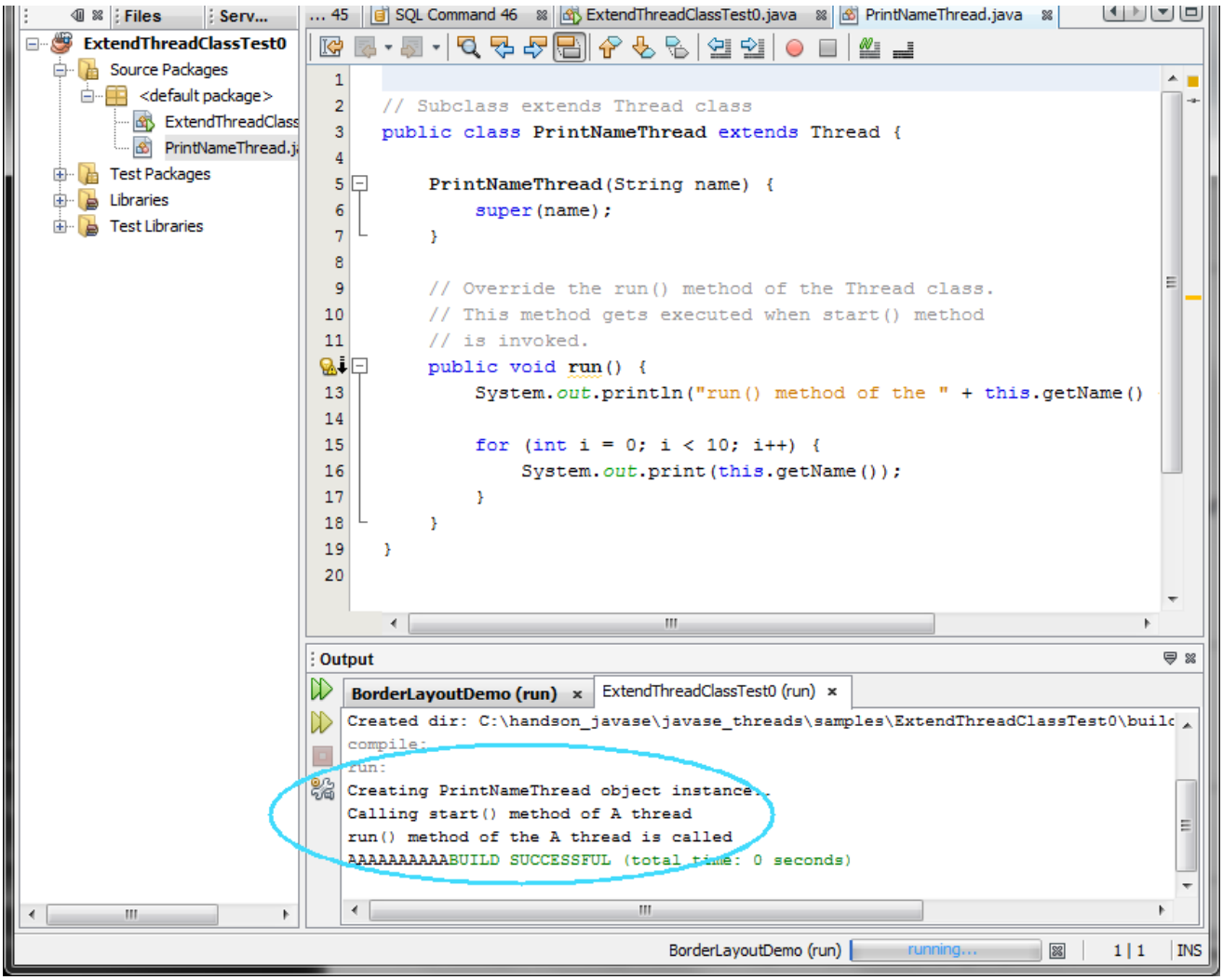
4. Build and run the project

- Right click **ExtendThreadClassTest0** project and select **Run**.
- Observe the result in the **Output** window. (Figure-1.13 below)

```
Creating PrintNameThread object instance..
Calling start() method of A thread
run() method of the A thread is called
AAAAAAAAAA
```

Figure-1.13: Result of running ExtendThreadClassTest0 application





5. Modify the **ExtendThreadClassTest0.java** as shown in Code-1.15 below. The code fragments that need to be added are highlighted in **bold and blue-colored** font.

```
public class ExtendThreadClassTest0 {

   public static void main(String args[]) {

      // Create object instance of a class that is subclass of Thread class
      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt1 =
            new PrintNameThread("A");

      // Start the thread by invoking start() method
      System.out.println("Calling start() method of " + pnt1.getName() + " thread");
      pnt1.start();

      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt2 =
            new PrintNameThread("B");
      System.out.println("Calling start() method of " + pnt2.getName() + " thread");
      pnt2.start();

      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt3 =
            new PrintNameThread("C");
      System.out.println("Calling start() method of " + pnt3.getName() + " thread");
      pnt3.start();
   }
}
```

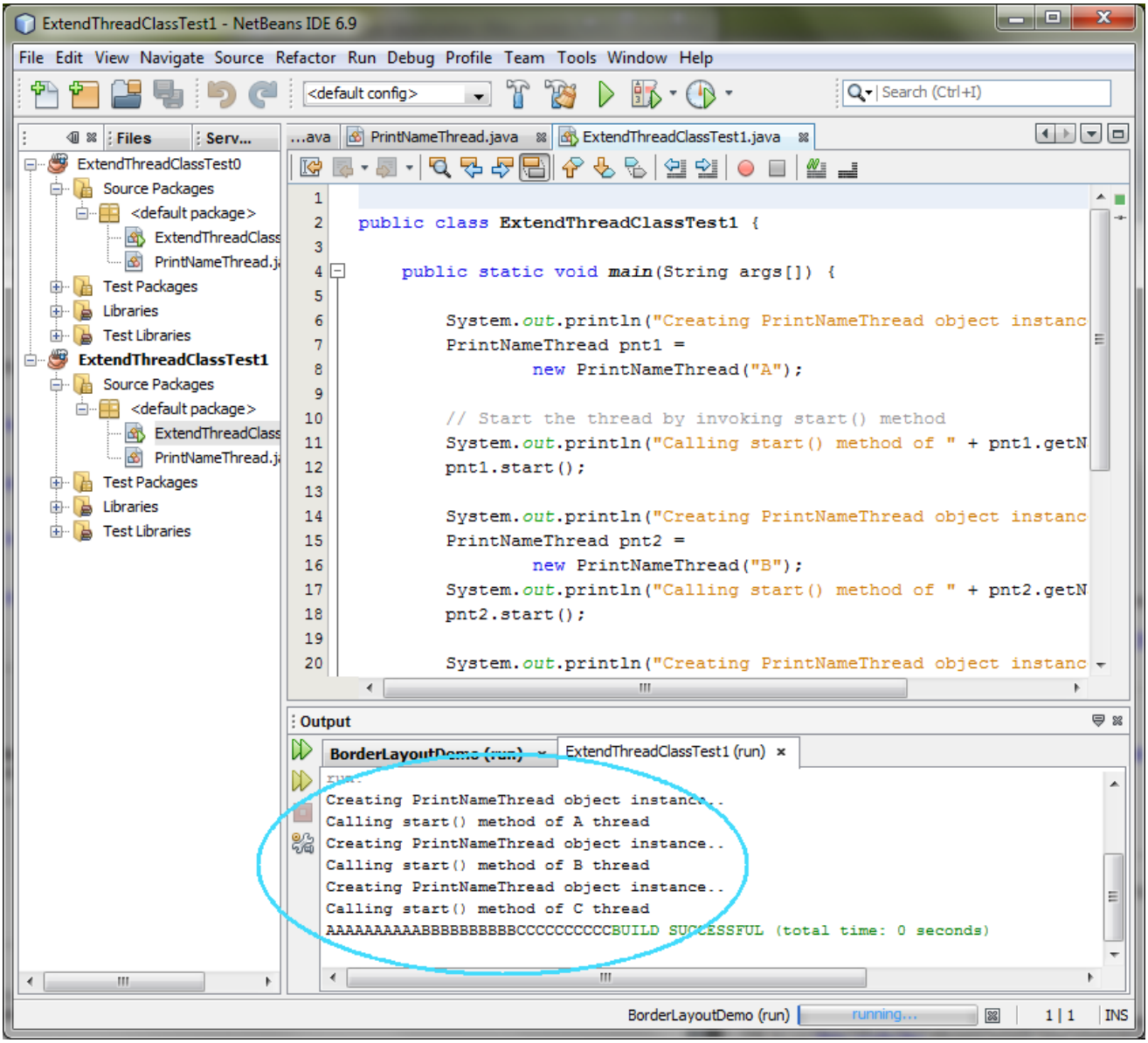
Code-1.15: Modified ExtendThreadClassTest0.java

6. Build and run the project

- Right click **ExtendThreadClassTest0** project and select **Run**.
- Observe the result in the **Output** window. (Figure-1.16 below)

```
Creating PrintNameThread object instance..
Calling start() method of A thread
Creating PrintNameThread object instance..
Calling start() method of B thread
AAAAAAAAAACreating PrintNameThread object instance..
BCalling start() method of C thread
BBBBBBBBBCCCCCCCCCC
```

Figure-1.16: Result



7. For your own exercise, modify **ExtendThreadClassTest0.java** as following. Build and run the application.

- Create and start another thread.
- Set the name of the thread as "MyOwn"

return to top of the exercise

(1.2) The start() method is in the constructor of the subclass

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ExtendThreadClassTest2** as project name.
- For **Create Main Class** field, type in **ExtendThreadClassTest2**.
- Click **Finish**.

- Observe that **ExtendThreadClassTest2** project appears and IDE generated **ExtendThreadClassTest2.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ExtendThreadClassTest2.java** as shown in Code-1.21 below.

```
public class ExtendThreadClassTest2 {

   public static void main(String args[]) {


      PrintNameThread pnt1 =
            new PrintNameThread("A");


      PrintNameThread pnt2 =
            new PrintNameThread("B");


      PrintNameThread pnt3 =
            new PrintNameThread("C");
```

```
    }
}
```

Code-1.21: ExtendThreadClassTest2.java

3. Write **PrintNameThread.java** as shown in Code-1.22 below. Note that the **start**() method is invoked as part of the constructor method of the **PrintNameThread** class.

```
public class PrintNameThread extends Thread {

    PrintNameThread(String name) {
        super(name);
        // start() method is inside the constructor of the subclass
        start();
    }

    public void run() {
        String name = getName();
        for (int i = 0; i < 10; i++) {
            System.out.print(name);
        }
    }
}
```

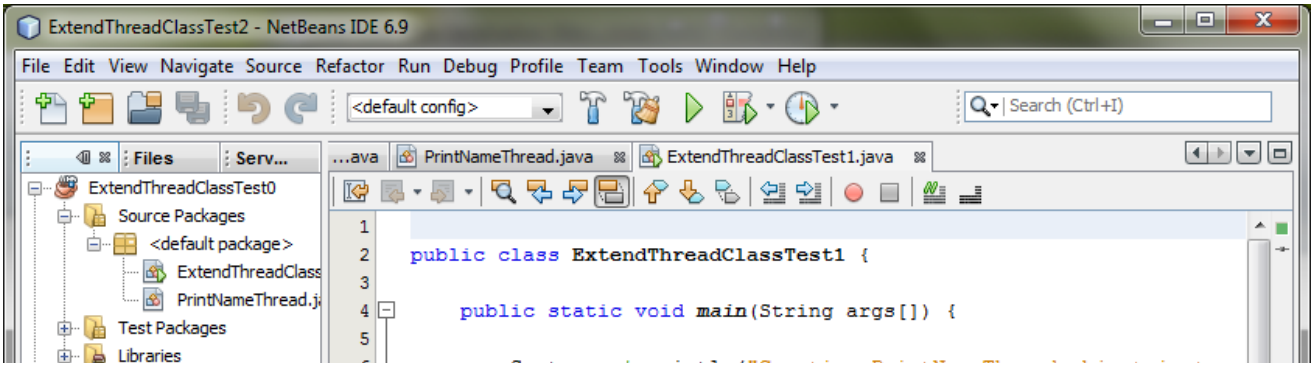
Code-1.22: PrintNameThread.java

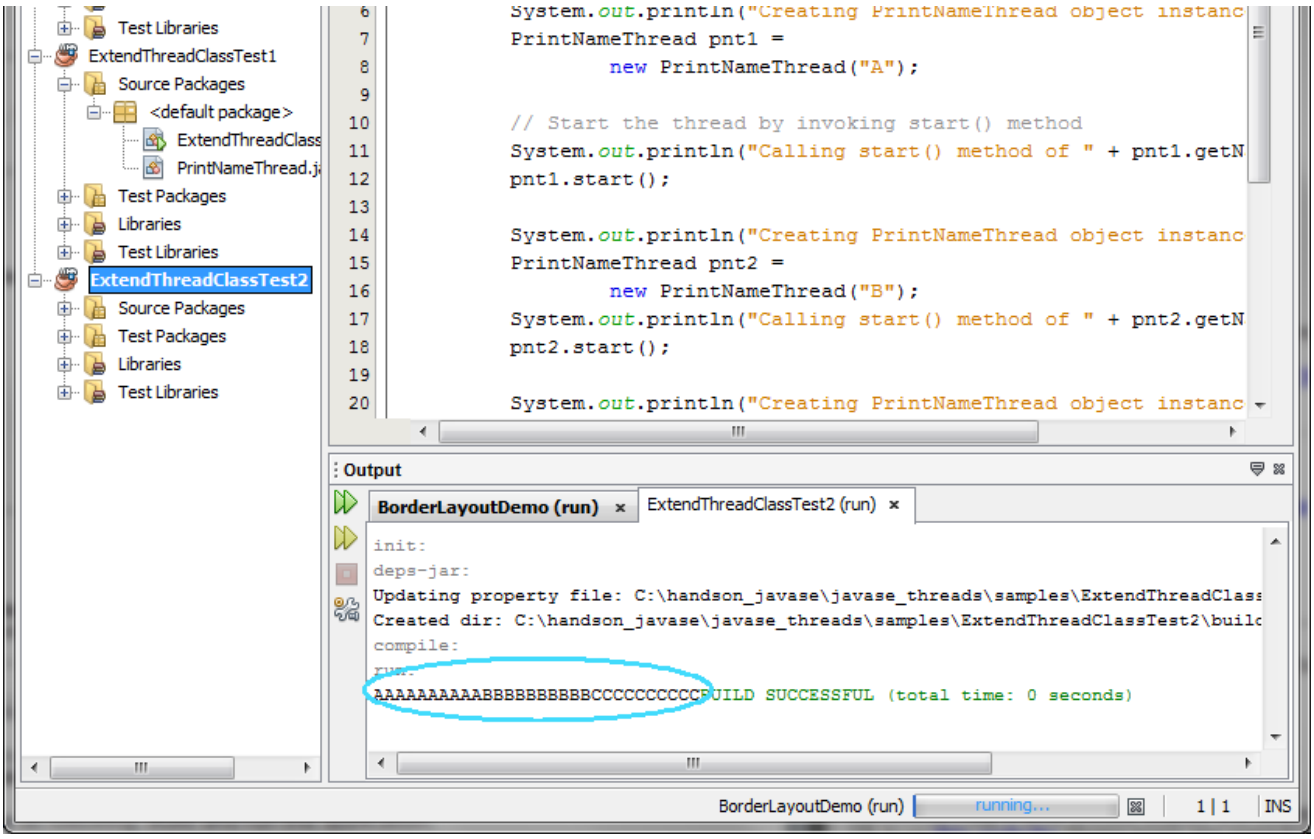
4. Build and run the project

- Right click **ExtendThreadClassTest2** project and select **Run**.
- Observe the result in the **Output** window. (Figure-1.23 below)

```
AAAAAAAAAABBBBBBBBBBCCCCCCCCCC
```

Figure-1.23: Result of running ExtendThreadClassTest2 application





5. For your own exercise, modify **ExtendThreadClassTest2.java** as following. Build and run the application.

- Create and start another thread.
- Set the name of the thread as "MyOwn"

return to top of the exercise

Summary

In this exercise, you have learned how to create and start a thread by extending Thread class.

return to the top

Exercise 2: Implement Runnable interface

In this exercise, you are going to create and start a thread by writing a class that implements Runnable interface.

1. **Create and start a thread by implementing Runnable interface - start() method is not in the constructor**
2. **Create and start a thread by implementing Runnable interface - start() method is in the constructor**

(2.1) Create and start a thread by implementing Runnable interface - start() method is not in the constructor

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **RunnableThreadTest1** as project name.
- For **Create Main Class** field, type in **RunnableThreadTest1**.
- Click **Finish**.

- Observe that **RunnableThreadTest1** project appears and IDE generated **RunnableThreadTest1.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **RunnableThreadTest1.java** as shown in Code-2.11 below. Study the code by paying special attention to the bold fonted parts. Note that the **start()** method needs to be invoked explicitly after an object instance of the **PrintNameRunnable** class is created.

```
public class RunnableThreadTest1 {

    public static void main(String args[]) {

        PrintNameRunnable pnt1 = new PrintNameRunnable("A");
        Thread t1 = new Thread(pnt1);
        t1.start();

        PrintNameRunnable pnt2 = new PrintNameRunnable("B");
        Thread t2 = new Thread(pnt2);
        t2.start();

        PrintNameRunnable pnt3 = new PrintNameRunnable("C");
        Thread t3 = new Thread(pnt3);
```

```
        Thread t3 = new Thread(print3);
        t3.start();

    }
}
```

Code-2.11: RunnableThreadTest1.java

3. Write **PrintNameRunnable.java** as shown in Code-2.12 below.

```
// The class implements Runnable interface
class PrintNameRunnable implements Runnable {

    String name;

    PrintNameRunnable(String name) {
        this.name = name;
    }

    // Implementation of the run() defined in the
    // Runnable interface.
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(name);
        }
    }
}
```

Code-2.12: PrintNameRunnable.java

4. Build and run the project

- Right click **RunnableThreadTest1** project and select **Run**.
- Observe the result in the **Output** window. (Figure-2.13 below)

```
ACBACBACBACBACABCABCABCABCABCB
```

Figure-2.13: Result of running RunnableThreadTest1 application

5. For your own exercise, do the following. Build and run the application.

- Create another class called **MyOwnRunnableClass** that implements Runnable interface
- **MyOwnRunnableClass** displays values 1 to 10 inside its run() method
- Modify **RunnableThreadTest1.java** to start 2 thread instances of **MyOwnRunnableClass**.

return to top of the exercise

(2.2) Create and start a thread by implementing Runnable interface - start() method is in the constructor

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **RunnableThreadTest2** as project name.
- For **Create Main Class** field, type in **RunnableThreadTest2**.
- Click **Finish**.

- Observe that **RunnableThreadTest2** project appears and IDE generated **RunnableThreadTest2.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **RunnableThreadTest2.java** as shown in Code-2.21 below. Study the code by paying special attention to the bold fonted parts.

```
public class RunnableThreadTest2 {

    public static void main(String args[]) {

        // Since the constructor of the PrintNameRunnable
        // object creates a Thread object and starts it,
        // there is no need to do it here.
        new PrintNameRunnable("A");

        new PrintNameRunnable("B");
        new PrintNameRunnable("C");
    }
}
```

Code-2.21: RunnableThreadTest2.java

3. Write PrintNameRunnable.java as shown in Code-2.22 below. Study the code by paying special attention to the bold fonted parts. Note that the start() method is in the constructor of the PrintNameRunnable class.

```
// The class implements Runnable interface
class PrintNameRunnable implements Runnable {

    Thread thread;

    PrintNameRunnable(String name) {
        thread = new Thread(this, name);
        thread.start();
    }

    // Implementation of the run() defined in the
```

```
    // Runnable interface.
    public void run() {
        String name = thread.getName();
        for (int i = 0; i < 10; i++) {
            System.out.print(name);
        }
    }
}
```

Code-2.22: PrintNameRunnable.java

4. Build and run the project

- Right click **RunnableThreadTest2** project and select **Run**.
- Observe the result in the **Output** window. (Figure-1.23 below)

```
ABCABCABCABCABCABCABCBACBACBAC
```

Figure-2.23: Result of running RunnableThreadTest2 application

return to top of the exercise

Summary

In this exercise, you have learned how to create a class that implements Runnable interface and starts a thread.

return to the top

Exercise 3: ThreadsGroup, View all threads, ThreadPriority

In this exercise, you are going to learn how to display information on a ThreadGroup, how to set a thread priority, and so on.

1. **Display threads of a ThreadGroup**
2. **Display all threads in the system**
3. **Set thread priority**

(3.1) Display threads of a ThreadGroup

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ThreadGroupTest** as project name.
- For **Create Main Class** field, type in **ThreadGroupTest**.
- Click **Finish**.

- Observe that **ThreadGroupTest** project appears and IDE generated **ThreadGroupTest.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ThreadGroupTest.java** as shown in Code-3.11 below. Study the code by paying special attention to the bold fonted parts.

```
public class ThreadGroupTest {

    public static void main (String[] args) {

        // Start three threads first.  They should belong
        // to a same ThreadsGroup.
        new SimpleThread("Boston").start();
        new SimpleThread("New York").start();
        new SimpleThread("Seoul").start();

        // Get ThreadGroup of the current thread and display
        // the number of active threads that belong to the
        // ThreadGroup.
        ThreadGroup group = Thread.currentThread().getThreadGroup();

        System.out.println("Number of active threads in this thread group = "
                        + group.activeCount());

        // Display the names of the threads in the current
        // ThreadGroup.
        Thread[] tarray = new Thread[10];
        int actualSize = group.enumerate(tarray);
        for (int i=0; i<actualSize;i++){
            System.out.println("Thread " + tarray[i].getName()
                        + " in thread group " + group.getName());
        }

    }
}
```

Code-3.11: ThreadGroupTest.java

3. Write **SimpleThread.java** as shown in Code-3.12 below.

```
public class SimpleThread extends Thread {

    public SimpleThread(String str) {
        super(str);
    }

    public void run() {
```

```
        for (int i = 0; i < 5; i++) {
            // System.out.format("%d %s%n", i, getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.format("DONE! %s%n", getName());
    }
}
```

Code-3.12: SimpleThread.java

4. Build and run the project

- Right click **ThreadGroupTest** project and select **Run**.
- Observe the result in the **Output** window. (Figure-3.13 below)

```
Number of active threads in this thread group = 4
Thread main in thread group main
Thread Boston in thread group main
Thread New York in thread group main
Thread Seoul in thread group main
DONE! Seoul
DONE! New York
DONE! Boston
```

Figure-3.13: Result of running ThreadGroupTest application

return to top of the exercise

5. For your own exercise, do the following. Build and run the application.

- Modify **ThreadGroupTest.java** to create another (4th) **SimpleThread** instance using your capital city of your country.

(3.2) Display all threads in the system

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **DisplayAllThreads** as project name.
- For **Create Main Class** field, type in **DisplayAllThreads**.
- Click **Finish**.

- Observe that **DisplayAllThreads** project appears and IDE generated **DisplayAllThreads.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **DisplayAllThreads.java** as shown in Code-3.21 below. Study the code by paying special attention to the bold fonted parts.

```
public class DisplayAllThreads {

    public static void main(String[] args) {

        // Start three threads first.  They should belong
        // to a same ThreadsGroup.
        new SimpleThread("Boston").start();
        new SimpleThread("New York").start();
        new SimpleThread("Seoul").start();

        Thread[] tarray = findAllThreads();

        for (int i=0; i<tarray.length;i++){
            System.out.println("Thread " + tarray[i].getName()
            + " in thread group " + tarray[i].getThreadGroup().getName());
        }

    }

    // Create an array of all threads in the system.
    public static Thread[] findAllThreads() {
        ThreadGroup group = Thread.currentThread().getThreadGroup();

        ThreadGroup topGroup = group;

        while (group != null) {
            topGroup = group;
            group = group.getParent();
        }

        int estimatedSize = topGroup.activeCount() * 2;
        Thread[] slackList = new Thread[estimatedSize];

        int actualSize = topGroup.enumerate(slackList);

        Thread[] list = new Thread[actualSize];
        System.arraycopy(slackList, 0, list, 0, actualSize);

        return list;
    }
}
```

Code-3.21: DisplayAllThreads.java

3. Write SimpleThread.java as shown in Code-3.22 below.

```
public class SimpleThread extends Thread {

   public SimpleThread(String str) {
      super(str);
   }

   public void run() {
      for (int i = 0; i < 5; i++) {
         // System.out.format("%d %s%n", i, getName());
         try {
            sleep((long)(Math.random() * 1000));
         } catch (InterruptedException e) {}
      }
      System.out.format("DONE! %s%n", getName());
   }
}
```

Code-3.22: SimpleThread.java

4. Build and run the project

- Right click **DisplayAllThreads** project and select **Run**.
- Observe the result in the **Output** window. (Figure-3.23 below)

```
Thread Reference Handler in thread group system
Thread Finalizer in thread group system
Thread Signal Dispatcher in thread group system
Thread main in thread group main
Thread Boston in thread group main
Thread New York in thread group main
Thread Seoul in thread group main
DONE! New York
DONE! Seoul
DONE! Boston
```

Figure-1.23: Result of running DisplayAllThreads application

5. For your own exercise, do the following. Build and run the application.

- Modify **DisplayAllThreads.java** to create another (4th) **SimpleThread** instance using your capital city of your country.

return to top of the exercise

(3.3) Set thread priority

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ThreadsPriority** as project name.
- For **Create Main Class** field, type in **ThreadsPriority**.
- Click **Finish**.

- Observe that **ThreadsPriority** project appears and IDE generated **ThreadsPriority.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ThreadsPriority.java** as shown in Code-3.31 below. Study the code by paying special attention to the bold fonted parts.

```
public class ThreadsPriority {

   public static void main(String[] args) {

      Thread t1 = new SimpleThread("Boston");
      t1.start();
      // Set the thread priority to 10(highest)
      t1.setPriority(10);

      Thread t2 = new SimpleThread("New York");
      t2.start();
      // Set the thread priority to 5
      t2.setPriority(5);

      Thread t3 = new SimpleThread("Seoul");
      t3.start();
      // Set the thread priority to 1
      t3.setPriority(1);

   }
}
```

Code-3.31: ThreadsPriority.java

3. Write SimpleThread.java as shown in Code-3.32 below.

```
public class SimpleThread extends Thread {

   public SimpleThread(String str) {
      super(str);
   }

   public void run() {
      for (int i = 0; i < 10; i++) {
```

```
        System.out.println(i + " " + getName()
                           + " Priority = " + getPriority());
      }
      System.out.println("Done! " + getName());
    }
}
```

Code-3.32: SimpleThread.java

4. Build and run the project

- Right click **ThreadsPriority** project and select **Run**.
- Observe the result in the **Output** window. (Figure-3.33 below)

```
0 Boston Priority = 10
0 Seoul Priority = 1
0 New York Priority = 5
1 Boston Priority = 10
1 Seoul Priority = 1
1 New York Priority = 5
2 Boston Priority = 10
2 Seoul Priority = 1
3 Boston Priority = 10
2 New York Priority = 5
4 Boston Priority = 10
3 New York Priority = 5
5 Boston Priority = 10
6 Boston Priority = 10
7 Boston Priority = 10
8 Boston Priority = 10
9 Boston Priority = 10
Done! Boston
4 New York Priority = 5
5 New York Priority = 5
6 New York Priority = 5
7 New York Priority = 5
8 New York Priority = 5
9 New York Priority = 5
Done! New York
3 Seoul Priority = 1
4 Seoul Priority = 1
5 Seoul Priority = 1
6 Seoul Priority = 1
7 Seoul Priority = 1
8 Seoul Priority = 1
9 Seoul Priority = 1
Done! Seoul
```

Figure-3.33: Result of running ThreadsPriority application

return to top of the exercise

Summary

In this exercise, you have learned how to retrieve information on a ThreadGroup.

return to the top

Exercise 4: Synchronization

In this exercise, you are going to exercise how to do synchronization among threads.

1. **Build and run a program in which threads are NOT synchronized**
2. **Build an run a program in which threads are synchronized through synchronized method**
3. **Build and run a program in which threads are synchronized through synchronized statement on a common object**

(4.1) Build and run a program in which threads are NOT synchronized

In this step, you are going to build an application that displays a result that is not desirable since threads are not synchronized.

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **UnsynchronizedExample** as project name.
- For **Create Main Class** field, type in **UnsynchronizedExample**.
- Click **Finish**.

- Observe that **UnsynchronizedExample** project appears and IDE generated **UnsynchronizedExample.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **UnsynchronizedExample.java** as shown in Code-4.11 below.

```
public class UnsynchronizedExample {

    public static void main(String[] args) {
        new PrintStringsThread("Hello ", "there.");
        new PrintStringsThread("How are ", "you?");
```

```
        new PrintStringsThread("How are ", "you?");
        new PrintStringsThread("Thank you ", "very much!");
    }

}
```

Code-4.11: UnsynchronizedExample.java

3. Write PrintStringsThread.java as shown in Code-4.12 below.

```
public class PrintStringsThread implements Runnable {

    Thread thread;
    String str1, str2;

    PrintStringsThread(String str1, String str2) {
        this.str1 = str1;
        this.str2 = str2;
        thread = new Thread(this);
        thread.start();
    }

    public void run() {
        TwoStrings.print(str1, str2);
    }

}
```

Code-4.12: PrintStringsThread.java

4. Write TwoStrings.java as shown in Code-4.13 below. Study the code by paying special attention to the bold fonted parts. Note that the print method is not synchronized.

```
public class TwoStrings {

    // This method is not synchronized
    static void print(String str1, String str2) {
        System.out.print(str1);
        try {
            Thread.sleep(500);
        } catch (InterruptedException ie) {
        }
        System.out.println(str2);
    }
}
```

Code-4.13: TwoStrings.java

5. Build and run the project

- Right click **UnsynchronizedExample** project and select **Run**.
- Observe the result in the **Output** window. (Figure-4.14 below)

```
Hello How are Thank you there.
very much!
you?
```

Figure-4.14: Result of running UnsynchronizedExample application

return to top of the exercise

(4.2) Build and run a program in which threads are synchronized through synchronized method

In this step, you are going to build an application that displays a desired result because the threads are synchronized.

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **SynchronizedExample1** as project name.
- For **Create Main Class** field, type in **SynchronizedExample1**.
- Click **Finish**.

- Observe that **SynchronizedExample1** project appears and IDE generated **SynchronizedExample1.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **SynchronizedExample1.java** as shown in Code-4.21 below.

```
public class SynchronizedExample1 {

    public static void main(String[] args) {
        new PrintStringsThread("Hello ", "there.");
        new PrintStringsThread("How are ", "you?");
        new PrintStringsThread("Thank you ", "very much!");
    }

}
```

Code-4.21: SynchronizedExample1.java

3. Write PrintStringsThread.java as shown in Code-4.22 below.

```
public class PrintStringsThread implements Runnable {
```

```
    Thread thread;
    String str1, str2;

    PrintStringsThread(String str1, String str2) {
        this.str1 = str1;
        this.str2 = str2;
        thread = new Thread(this);
        thread.start();
    }

    public void run() {
        TwoStrings.print(str1, str2);
    }

}
```

Code-4.22: PrintStringsThread.java

4. Write TwoStrings.java as shown in Code-4.23 below. Study the code by paying special attention to the bold fonted parts.

```
public class TwoStrings {

    // This method is now synchronized
    synchronized static void print(String str1, String str2) {
        System.out.print(str1);
        try {
            Thread.sleep(500);
        } catch (InterruptedException ie) {
        }
        System.out.println(str2);
    }
}
```

Code-4.23: TwoStrings.java

5. Build and run the project

- Right click **SynchronizedExample1** project and select **Run**.
- Observe the result in the **Output** window. (Figure-4.24 below)

```
How are you?
Thank you very much!
Hello there.
```

Figure-4.24: Result of running UnSynchronizedExample1 application

return to top of the exercise

(4.3) Build and run a program in which threads are synchronized through synchronized statement on common object

In this step, you are going to build another application that displays a desired result because the threads are synchronized.

1. Create a new NetBeans project
   - Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
   - Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
   - Click **Next**.
   - Under **Name and Location** pane, for the **Project Name** field, type in **SynchronizedExample2** as project name.
   - For **Create Main Class** field, type in **SynchronizedExample2**.
   - Click **Finish**.
   - Observe that **SynchronizedExample2** project appears and IDE generated **SynchronizedExample2.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **SynchronizedExample2.java** as shown in Code-4.31 below.

```
public class SynchronizedExample2 {

    public static void main(String[] args) {

        TwoStrings ts = new TwoStrings();

        new PrintStringsThread("Hello ", "there.", ts);
        new PrintStringsThread("How are ", "you?", ts);
        new PrintStringsThread("Thank you ", "very much!", ts);
    }

}
```

Code-4.31: SynchronizedExample2.java

3. Write PrintStringsThread.java as shown in Code-4.32 below. Study the code by paying special attention to the bold fonted parts.

```
public class PrintStringsThread implements Runnable {

    Thread thread;
    String str1, str2;
    TwoStrings ts;

    PrintStringsThread(String str1, String str2,
                       TwoStrings ts) {
        this.str1 = str1;
        this.str2 = str2;
        this.ts = ts;
        thread = new Thread(this);
        thread.start();
```

```
    }

    public void run() {
        // Synchronize over TwoString object
        synchronized (ts) {
            ts.print(str1, str2);
        }
    }
}
```

Code-4.32: PrintStringsThread.java

4. Write TwoStrings.java as shown in Code-4.33 below.

```
public class TwoStrings {

    static void print(String str1, String str2) {
        System.out.print(str1);
        try {
            Thread.sleep(500);
        } catch (InterruptedException ie) {
        }
        System.out.println(str2);
    }
}
```

Code-4.33: TwoStrings.java

5. Build and run the project

- Right click **SynchronizedExample2** project and select **Run**.
- Observe the result in the **Output** window. (Figure-4.34 below)

```
How are you?
Thank you very much!
Hello there.
```

Figure-4.34: Result of running UnSynchronizedExample2 application

return to top of the exercise

Summary

In this exercise, you have learned how to use synchronization.

return to the top

Exercise 5: Inter-thread communication

1. **Producer-Consumer without inter-thread communication**
2. **Producer-Consumer with inter-thread communication**

(5.1) Producer-Consumer without inter-thread communication

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ProducerConsumerUnsynchronized** as project name.
- For **Create Main Class** field, type in **ProducerConsumerUnsynchronized**.
- Click **Finish**.

- Observe that **ProducerConsumerUnsynchronized** project appears and IDE generated **ProducerConsumerUnsynchronized.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ProducerConsumerUnsynchronized.java** as shown in Code-5.11 below.

```
public class ProducerConsumerUnsynchronized {

    public static void main(String[] args) {

        CubbyHole c = new CubbyHole();

        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```

Code-5.11: ProducerConsumerUnsynchronized.java

3. Write **CubbyHole.java** as shown in Code-5.12 below. Study the code by paying special attention to the bold fonted parts.

```
// Unsynchronized CubbyHole.
//
// Results are unpredictable; a number may be read before a number has
// been produced or multiple numbers may be produced with only one or
// two being read adding synchronization ensures that a number is first
// produced, then read in the correct order.

public class CubbyHole {
```

```
    private int contents;
    private boolean available = false;

    public int get() {
        available = false;
        return contents;
    }

    public void put(int value) {
        contents = value;
        available = true;
    }
}
```

Code-5.12: CubbyHole.java

4. Write **Producer.java** as shown in Code-5.13 below.

```
public class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number
                               + " put: " + i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Code-5.13: Producer.java

5. Write **Consumer.java** as shown in Code-5.14 below.

```
public class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;


    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number
                               + " got: " + value);
        }
    }
}
```

Code-5.14: Consumer.java

6. Build and run the project

- Right click **ProducerConsumerUnsynchronized** project and select **Run**.
- Observe the result in the **Output** window. (Figure-5.15 below)

```
Producer #1 put: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Consumer #1 got: 0
Producer #1 put: 1
Producer #1 put: 2
Producer #1 put: 3
Producer #1 put: 4
Producer #1 put: 5
Producer #1 put: 6
Producer #1 put: 7
Producer #1 put: 8
Producer #1 put: 9
```

Figure-5.15: Result of running ProducerConsumerUnsynchronized application

return to top of the exercise

(5.2) Producer-Consumer with inter-thread communication

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **ProducerConsumerSynchronized** as project name.
- For **Create Main Class** field, type in **ProducerConsumerSynchronized**.
- Click **Finish**.

- Observe that **ProducerConsumerSynchronized** project appears and IDE generated **ProducerConsumerSynchronized.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ProducerConsumerSynchronized.java** as shown in Code-5.21 below.

```
public class ProducerConsumerSynchronized {

    public static void main(String[] args) {

        CubbyHole c = new CubbyHole();

        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```

Code-5.21: ProducerConsumerSynchronized.java

3. Write **CubbyHole.java** as shown in Code-5.22 below. Study the code by paying special attention to the bold fonted parts.

```
public class CubbyHole {

    private int contents;
    private boolean available = false;

    public synchronized int get(int who) {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        available = false;
        System.out.format("Consumer %d got: %d%n", who, contents);
        notifyAll();
        return contents;
    }

    public synchronized void put(int who, int value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        System.out.format("Producer %d put: %d%n", who, contents);
        notifyAll();
    }
}
```

Code-5.22: CubbyHole.java

4. Write **Producer.java** as shown in Code-5.23 below.

```
public class Producer extends Thread {

    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(number, i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Code-5.23: Producer.java

5. Write **Consumer.java** as shown in Code-5.24 below.

```
public class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
```

```
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get(number);
        }
    }
}
```

Code-5.24: Consumer.java

6. Build and run the project

- Right click **ProducerConsumerSynchronized** project and select **Run**.
- Observe the result in the **Output** window. (Figure-5.25 below)

```
Producer 1 put: 0
Consumer 1 got: 0
Producer 1 put: 1
Consumer 1 got: 1
Producer 1 put: 2
Consumer 1 got: 2
Producer 1 put: 3
Consumer 1 got: 3
Producer 1 put: 4
Consumer 1 got: 4
Producer 1 put: 5
Consumer 1 got: 5
Producer 1 put: 6
Consumer 1 got: 6
Producer 1 put: 7
Consumer 1 got: 7
Producer 1 put: 8
Consumer 1 got: 8
Producer 1 put: 9
Consumer 1 got: 9
```

Figure-5.25: Result of running ProducerConsumerSynchronized application

return to top of the exercise

Summary

In this exercise, you have learned how to perform inter-thread commmunication by the usage of wait(), notify(), and notifyAll() methods.

return to the top

Exercise 6: Timer and TimerTask

In this exercise, you will learn how to use Timer and TimerTask to schedule a single or repeating task.

1. **Schedule one-time task**
2. **Schedule repeating task**

(6.1) Schedule one-time task

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **TimerReminder** as project name.
- For **Create Main Class** field, type in **TimerReminder**.
- Click **Finish**.

- Observe that **TimerReminder** project appears and IDE generated **TimerReminder.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **TimerReminder.java** as shown in Code-6.11 below. Study the code by paying special attention to the bold fonted parts.

```
import java.util.Timer;
import java.util.TimerTask;

/**
 * Simple demo that uses java.util.Timer to schedule a task to execute
 * once 5 seconds have passed.
 */

public class TimerReminder {

    Timer timer;

    public TimerReminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    }

    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Time's up!");
```

```
            timer.cancel(); //Terminate the timer thread
        }
    }

    public static void main(String args[]) {
        System.out.println("About to schedule Reminder task in 5 seconds");
        new TimerReminder(5);
        System.out.println("Task scheduled.");
    }
}
```

Code-6.11: TimerReminder.java

3. Build and run the project

- Right click **TimerReminder** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.12 below)

```
About to schedule Reminder task in 5 seconds
Task scheduled.
Time's up!
```

Figure-6.12: Result of running UnTimerReminder application

return to top of the exercise

(6.2) Schedule a repeating task

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **AnnoyingBeep** as project name.
- For **Create Main Class** field, type in **AnnoyingBeep**.
- Click **Finish**.

- Observe that **AnnoyingBeep** project appears and IDE generated **AnnoyingBeep.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **AnnoyingBeep.java** as shown in Code-6.21 below. Study the code by paying special attention to the bold fonted parts.

```
import java.util.Timer;
import java.util.TimerTask;
import java.awt.Toolkit;

/**
 * Schedule a task that executes once every second.
 * Beep every second.
 */

public class AnnoyingBeep {
    Toolkit toolkit;
    Timer timer;

    public AnnoyingBeep() {
        toolkit = Toolkit.getDefaultToolkit();
        timer = new Timer();
        timer.schedule(new RemindTask(),
                0,        //initial delay
                1*1000);  //subsequent rate
    }

    class RemindTask extends TimerTask {
        int numWarningBeeps = 3;

        public void run() {
            if (numWarningBeeps > 0) {
                toolkit.beep();
                System.out.format("Beep!%n");
                numWarningBeeps--;
            } else {
                toolkit.beep();
                System.out.format("Time's up!%n");
                //timer.cancel(); //Not necessary because we call System.exit
                System.exit(0);   //Stops the AWT thread (and everything else)
            }
        }
    }

    public static void main(String args[]) {
        System.out.format("About to schedule task.%n");
        new AnnoyingBeep();
        System.out.format("Task scheduled.%n");
    }
}
```

Code-6.21: AnnoyingBeep.java

3 Build and run the project

- Right click **AnnoyingBeep** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.22 below)

```
About to schedule task.
Task scheduled.
Beep!
Beep!
Beep!
```

```
Time's up!
```

Figure-6.22: Result

return to top of the exercise

Summary

In this exercise, you have learned how to use Timer and TimerTask classes to schedule one-time or repeating tasks.

return to the top

Homework

1. The homework is to create a new NetBeans project called **MyRunnableProject** as following.

- Create a class called **MyCurrentDate** that implements **Runnable** interface.
- The **MyCurrentDate** class displays the current date and time 10 times, with 100 milli seconds interval - use **sleep()** method for this interval.
- Create a class called **MyMain**, which contans **main()** method, in which 3 instances of **MyCurrentDate** threads are being run.