Dimas Sheldon
1415002

February 24, 2017

EKPL II: Multithreading

1. Test Result

1.1. Procedure 1

a. Program

```
package EKPL2.Lab.February23_Multithreading.Procedure1;

public class PrintChar {
  private char charToPrint;
  private int times;

  PrintChar(char c, int t) {
    charToPrint = c;
    times = t;
  }

  public void print() {
    for (int i = 0; i < times; i++) {
      System.out.print(charToPrint + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure1;

public class PrintNum {
  private int lastNum;

  PrintNum(int n) {
    lastNum = n;
  }

  public void print() {
    for (int i = 0; i < lastNum; i++) {
      System.out.print("" + i + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure1;

public class Test {
  public static void main(String[] args) {
    PrintChar printA = new PrintChar('a', 5);
    PrintChar printB = new PrintChar('b', 5);
    PrintNum printN = new PrintNum(5);
    printA.print();
    printB.print();
    printN.print();
  }
}
```

b. Output

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
a a a a a b b b b b 0 1 2 3 4
Process finished with exit code 0
```

1.2. Procedure 2

a. Program

```
package EKPL2.Lab.February23_Multithreading.Procedure2;

public class PrintChar extends Thread {
  private char charToPrint;
  private int times;
  private int DELAY = 500;

  public PrintChar(char c, int t) {
    charToPrint = c;
    times = t;
  }

  @Override
  public void run() {
    for (int i = 0; i < times; i++) {
      try {
        sleep(DELAY);
      } catch (InterruptedException ie){
      }
      System.out.print(charToPrint + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure2;

public class PrintNum extends Thread {
  private int lastNum;
  private int DELAY = 250;

  PrintNum(int n) {
    lastNum = n;
  }

  @Override
  public void run() {
    for (int i = 0; i < lastNum; i++) {
      try {
        sleep(DELAY);
      } catch (InterruptedException ie) {
      }
      System.out.print("" + i + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure2;

public class ThreadRunner {
  public static void main(String[] args) {
    PrintChar printA = new PrintChar('A', 15);
    PrintChar printB = new PrintChar('B', 15);
    PrintNum printN = new PrintNum(15);

    printA.start();
    printB.start();
    printN.start();
  }
}
```

b. Output

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
0 A B 1 2 A B 3 4 A B 5 6 A 7 B 8 A B 9 10 A 11
 B 12 A B 13 14 A B A B A B A B A B A B A B A B
Process finished with exit code 0
```

1.3. Procedure 3

a. Program

```
package EKPL2.Lab.February23_Multithreading.Procedure3;

public class PrintChar implements Runnable {
  private char charToPrint;
  private int times;
  private int DELAY = 250;

  public PrintChar(char c, int t) {
    charToPrint = c;
    times = t;
  }

  @Override
  public void run() {
    for (int i = 0; i < times; i++) {
      try {
        Thread.currentThread().sleep(DELAY);
      } catch (InterruptedException ie) {
      }
      System.out.print(charToPrint + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure3;

public class PrintNum implements Runnable {
  private int lastNum;
  private int DELAY = 500;

  PrintNum(int n) {
    lastNum = n;
  }

  @Override
  public void run() {
    for (int i = 0; i < lastNum; i++) {
      try {
        Thread.currentThread().sleep(DELAY);
      } catch (InterruptedException ie) {
      }
      System.out.print("" + i + " ");
    }
  }
}
```

```
package EKPL2.Lab.February23_Multithreading.Procedure3;

public class ThreadRunnerRunnable {
  public static void main(String[] args) {
```

```
    Thread printA = new Thread(new PrintChar('A', 15));
    Thread printB = new Thread(new PrintChar('B', 15));
    Thread printN = new Thread(new PrintNum(15));

    printA.start();
    printB.start();
    printN.start();
  }
}
```

b. Output

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
B A A B 0 B A 1 A B A B 2 A B A B 3 B A B A 4 B A
 B A 5 B A B A 6 A B B A 7 8 9 10 11 12 13 14
Process finished with exit code 0
```

1.4. Procedure 4

```
package EKPL2.Lab.February23_Multithreading.Procedure4;

public class PiggyBank {
  private int balance = 0;

  public int getBalance() {
    return balance;
  }

  public void setBalance(int balance) {
    this.balance = balance;
  }
}
```

a. Program 4a

```
package EKPL2.Lab.February23_Multithreading.Procedure4;

public class PiggyBankApp {
  private PiggyBank bank = new PiggyBank();
  private Thread[] threads = new Thread[100];

  public PiggyBankApp() {
    ThreadGroup g = new ThreadGroup("group");
    boolean done = false;

    for (int i = 0; i < 100; i++) {
      threads[i] = new Thread(g, new AddPennyThread(), "t");
      threads[i].start();
    }


    while (!done) {
      if (g.activeCount() == 0) {
        done = true;
      }
    }
  }

  class AddPennyThread extends Thread {
    public void run() {
```

```
      int newBalance = bank.getBalance() + 1000;
      System.out.println("Tabung 1000 => Balance = " + newBalance);

      try {
        Thread.sleep(5);
      } catch (InterruptedException ie) {
        System.out.println(ie);
      }

      bank.setBalance(newBalance);
    }
  }

  public static void main(String[] args) {
    PiggyBankApp test = new PiggyBankApp();
    System.out.println("What is balance? " + test.bank.getBalance());
  }
}
```

b. Output 4a

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 5000
Tabung 1000 => Balance = 5000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 4000
Tabung 1000 => Balance = 5000
Tabung 1000 => Balance = 6000
Tabung 1000 => Balance = 6000
What is balance? 6000

Process finished with exit code 0
```

c. Program 4b

```
package EKPL2.Lab.February23_Multithreading.Procedure4;

public class PiggyBankAppSync {
  private PiggyBank syncBank = new PiggyBank();
  private Thread[] threads = new Thread[100];

  public PiggyBankAppSync() {
    ThreadGroup g = new ThreadGroup("group");
    boolean done = false;

    for (int i = 0; i < 100; i++) {
      threads[i] = new Thread(g, new AddPennyThread(), "t");
```

```
      threads[i].start();
    }

    while (!done) {
      if (g.activeCount() == 0) {
        done = true;
      }
    }
  }

  private static synchronized void addPenny(PiggyBank bank) {
    int newBalance = bank.getBalance() + 1000;
    try {
      Thread.sleep(5);
    } catch (InterruptedException ie) {
    }
    bank.setBalance(newBalance);
  }

  class AddPennyThread extends Thread {
    public void run() {
      int newBalance = syncBank.getBalance() + 1000;
      System.out.println("Tabung 1000 => Balance = " + newBalance);
      addPenny(syncBank);
    }
  }
  public static void main(String[] args) {
    PiggyBankAppSync test = new PiggyBankAppSync();
    System.out.println("What is balance? " +
test.syncBank.getBalance());
  }
}
```

d. Output 4b

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 1000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
• • •
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 2000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
Tabung 1000 => Balance = 3000
What is balance? 100000

Process finished with exit code 0
```

2. Analysis

In procedure 1, the process is a general form of algorithm by doing one process to another from the top. In this case, it will create object printA, printB, and printN, then the program will execute printA.method() first, resulting of (a a a a a). After printA.method is finished, the process is continued by executing printB.method(), resulting (b b b b b), and so on - until no process to execute anymore.

In procedure 2, each object is created from a class that extending Thread class which will executes the run() method from each object itself. In this case, We have 3 objects of Thread class, printA, printB, and printN. When the program is running, each object will execute their run() method, and each of them will pausing their work in specified time to let another thread to run.

Differ with procedure 2, in procedure 3 each object is created from Thread class, but it's using the Runnable object as its parameter to specify the overridden run() method. When the program is running, the process is nearly the same with procedure 2, one of the difference is in procedure 3, it's executing the Thread class' methods, not a class that inheriting from it.

In procedure 4, we have the unsynchronized and synchronized class method which has some threads using the same resource. As what we have learned, if some tasks with the same resource working at the same time, it will cause 'collision' or error in data interpreting – as shown in procedure 4a, resulting of wrong answer. This may happen caused by task[i] re-created what task[j] have done before. To solve this, in procedure 4b, we add synchronizing method that used to determine whether some tasks have to run or not. After adding this synchronize method, we get the desired result.

3. Conclusion

In Java, multiple inheritance is not allowed. If we are not making any modification on Thread class, we should use Runnable interface. It's important to manage the program to be as effective and efficient as possible. Rather than creating multiple objects from extended class, it's better to use an object from one class that can implement multiple tasks. This consideration will give us some advantage, like memory efficiency.

In making a program with more than one thread using same resource, we need to synchronize them so there will be no **lack of resource** or threads conflict – that can cause wrong interpretation.