Multithreading 2

OODP, 2022

```
class PingPong extends Thread {
 String word;
 int delay;
 PingPong(String whatToSay,
                 int delayTime)
{
   word = whatToSay; delay = delayTime;}
public void run() {
  try {
    for (;;) {
      System.out.print(word + " ");
      sleep(delay);}
   } catch (InterruptedException e) {
      return;}}
 public static void main(String[] args) {
     new PingPong("ping", 33).start();
     new PingPong("PONG", 100).start();
         }
    }
```

>java PingPong

Run and observe the output behavior!

Consider sleep time of two threads.

```
public class Simultaneous {
    public static void main (String[] args) {
        Soda one = new Soda ("Coke");
        Soda two = new Soda ("Pepsi");
        Soda three = new Soda ("Diet Coke");
        one.start();
        two.start();
        three.start();
    }
}
```

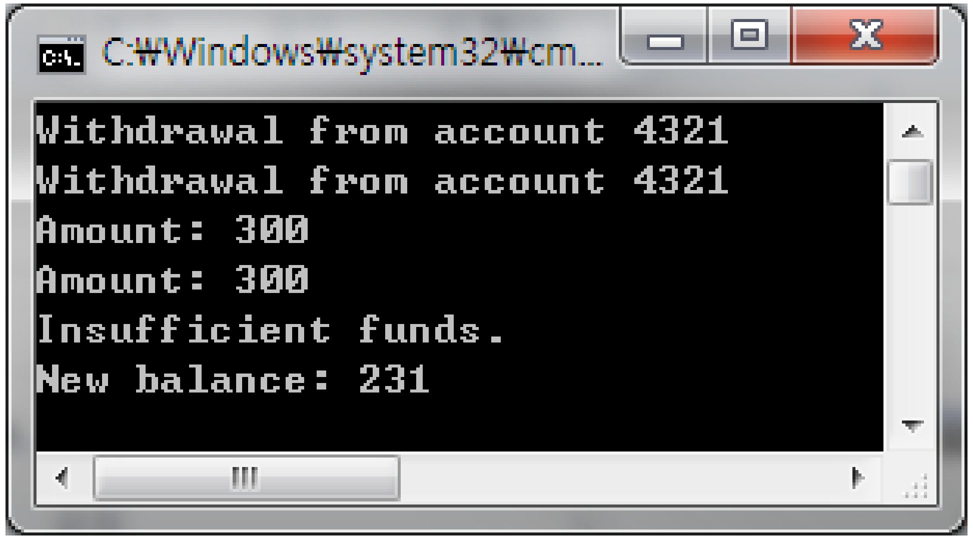
```
class Soda extends Thread {
    private String name;
    Soda (String str) {
        name = str;
    }  // constructor Soda
    public void run() {
        for (int count = 0; count < 10; count++)
            System.out.println (name);
    }
}
```

Observe the running outputs with
3 threads running concurrently…

```
public class ATM_Accounts {
   public static void main (String[] args) {
      Savings_Account savings = new
               Savings_Account (4321, 531);
      ATM west_branch = new ATM (savings);
      ATM east_branch = new ATM (savings);
      west_branch.start();
      east_branch.start();
   }
}
```

```
class Savings_Account {
    protected int account;
    protected int balance;
    public Savings_Account (int account_num, int initial)
{
        account = account_num;
        balance = initial;
    }
public synchronized boolean withdrawal (int amount)
{
        boolean result = false;
…
public synchronized boolean withdrawal (int amount) {
        boolean result = false;
        System.out.println ("Withdrawal from account " + account);
        System.out.println ("Amount: " + amount);
        if (amount <= balance) {
            balance -= amount;
            System.out.println ("New balance: " + balance);
            result = true;
        } else
            System.out.println ("Insufficient funds.");
        System.out.println();
        return result;
    }
}
```

```
class ATM extends Thread {
    Savings_Account account;
    public ATM (Savings_Account savings) {
        account = savings;
    }
    public void run () {
        account.withdrawal (300);
    }}
```



```
Withdrawal from account 4321
Withdrawal from account 4321
Amount: 300
Amount: 300
Insufficient funds.
New balance: 231
```

```
public class Counter1 extends Thread {
    protected int count;
    protected int inc;
    protected int delay;
    public Counter1( int  init,  int  inc,  int  delay ) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }
```

```
 public void run() {
        try {
            for (;;) {
                System.out.print(count + " ");
                count += inc;
                sleep(delay);
            }
        } catch (InterruptedException e) {}
    }
    public static void main(String[] args) {
        new Counter1(0, 1, 33).start();
        new Counter1(0, -1, 100).start();
    }
}
```

Observe the running output behavior.

```java
public class Counter2 implements Runnable {
    protected int count;
    protected int inc;
    protected int delay;
    public Counter2(int init, int inc, int delay) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }
```

```
 public void run() {
        try {
            for (;;) {
                System.out.print(count + " ");
                count += inc;
                Thread.sleep(delay);
            }
        } catch (InterruptedException e) {}
    }
    public static void main(String[] args) {
        new Thread(new Counter2(0, 1, 33)).start();
        new Thread(new Counter2(0, -1, 100)).start();
    }
}
```

Run the program and observe the output.