训练部分

基础部分

提交方式：

请提交`basic.json`文件至第二单元个人训练仓库，格式与第一单元相同：

```
{
    "1":"编号（1）处输出结果",
    "2":"编号（2）处输出结果",
    ...
}
```

task 0 基础填空

- 实现多线程的两种方法是（1）和（2）。启动线程时调用（3）方法。
- 使用（4）关键字可以使任何时候只允许一个线程获得访问/执行权限。
- （5）方法随机从Wait Set中选择一个线程唤醒，（6）方法将Wait Set中所有线程唤醒。
- 为了使对象只创建一个实例，并且提供一个全局的访问点，可以使用设计模式中的（7）。

task 1 补全程序

```
public class Process1 /* to complete (8) */ {
    @Override
    public void run() {
        super.run();
    }
}

public class Process2 implements Runnable {
    @Override
    public void run() {
        System.out.println("run");
    }
}

public class MainClass {
    public static void main(String[] args) {
        Process1 process1 = new Process1();
        process1.start();
        Process2 process2 = new Process2();
        /* to complete(9), make process start*/
```

```
    }
}
```

task 2 分析计算结果

请分析staticVar、classVar、localVar的值

（分析每一个线程结束时每个量的值，本题为思考题不必作答）

```
public class Process1 extends Thread {
    static int staticVar = 0;
    int classVar = 0;

    @Override
    public void run() {
        add();
        add();
    }

    private void add() {
        int localVar = 0;
        staticVar++;
        classVar++;
        localVar++;
    }
}

public class MainClass {
    public static void main(String[] args) {
        Process1 process1 = new Process1();
        process1.start();
        Process1 process11 = new Process1();
        process11.start();
    }
}
```

task3 分析是否能正确结束

```
public class Process1 extends Thread {
    private final Somethings somethings;

    public Process1(Somethings somethings) {
        this.somethings = somethings;
    }
```

```
    @Override
    public void run() {
        while (true) {
            synchronized (somethings) {
                if (somethings.getNumber() > 5) {
                    return;
                }
                try {
                    somethings.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                if (somethings.getNumber() % 2 == 0) {
                    somethings.setNumber(somethings.getNumber() + 1);
                }
                somethings.notifyAll();
            }
            try {
                sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }
}

public class Process2 extends Thread {
    private final Somethings somethings;

    public Process2(Somethings somethings) {
        this.somethings = somethings;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (somethings) {
                if (somethings.getNumber() > 5) {
                    return;
                }
                if (somethings.getNumber() % 2 == 1) {
                    somethings.setNumber(somethings.getNumber() + 1);
                }
                somethings.notifyAll();
            }
            try {
                sleep(150);
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            synchronized (somethings) {
                try {
                    somethings.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

public class Somethings {
    private int number;

    public Somethings() {
        number = 0;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}

public class MainClass {
    public static void main(String[] args) {
        Somethings somethings = new Somethings();
        Process1 process1 = new Process1(somethings);
        process1.start();
        Process2 process2 = new Process2(somethings);
        process2.start();
    }
}
```

请选择程序没能结束的原因（10）

A. Process1占用锁，导致Process2始终无法开始执行。

B. Process2占用锁，导致Process1始终无法开始执行。

C. Process1处于sleep状态/未处于wait状态时Process2使用notifyAll导致本次唤醒无效，或Process2处于sleep状态/未处于wait状态时Process1使用notifyAll导致本次唤醒无效，导致最终两个线程都处于wait状态无法结束。

D.程序正确可以正常结束。

提升部分

提交方式：

请提交完成后的代码至第二单元个人训练仓库（和json文件放至同一仓库即可）

task4 生产者消费者模式

实现一个简单的生产者消费者模型，要求如下：

托盘容量为1。

Producer生产10个货物，每生产一个货物后会立刻尝试放入，放入成功前不会继续生产，货物按照从1-10编号。

Producer在向托盘中成功放入货物后需要sleep 0-100ms，可用sleep((int)(Math.random() * 100))实现。

Consumer只能取托盘中现有的货物。

Producer在向托盘中放入货物时需输出 "Producer put:" + 货物编号

Consumer在从托盘中取出货物时需输出 "Consumer get:" + 货物编号

提示：

- 请确保你的程序能够正常结束
- 请用多线程的方法解题，不允许直接输出结果

task5 观察者模式

观察者接口

```
public interface Observer {
    void update(String msg); //在该方法中打印msg
}
```

被观察者接口

```
public interface Observerable {
    void addObserver(Observer observer);

    void removeObserver(Observer observer);

    void notifyObserver(String msg); //打印msg并调用每个注册过的观察者的update方法
}
```

请创建两个类实现上述两个接口并使得下面main方法可以运行，且运行结果如下

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

```
    Server server = new Server(); //该类实现Observerable接口

    Observer user1 = new User("user1"); //该类实现Observer接口
    Observer user2 = new User("user2"); //该类实现Observer接口

    server.addObserver(user1);
    server.addObserver(user2);
    server.notifyObserver("北航的OO课是世界上最好的OO课！");

    server.removeObserver(user2);
    server.notifyObserver("Java是世界上最好用的语言。");
}
```

运行结果：

> server: 北航的OO课是世界上最好的OO课！
>
> user1: 北航的OO课是世界上最好的OO课！
>
> user2: 北航的OO课是世界上最好的OO课！
>
> server: Java是世界上最好用的语言。
>
> user1: Java是世界上最好用的语言。