Chapter 11-Instep Demo

Demo1:

Creating Thread by extending (inheriting) the Thread Class

Program to create two threads to Print Odd number and Even number at the same time by extending thread class.

```
package chapter11;

/**
 * 奇数线程类 - Odd Number Thread Class
 * 任务：创建一个线程来打印 1 到 20 之间的奇数
 * Task: Create a thread to print odd numbers between 1 and 20
 */
class OddNumber extends Thread {
    /**
     * 步骤 1：重写 run 方法，定义线程执行的任务
     * Step 1: Override the run method to define the thread's execution task
     */
    @Override
    public void run() {
        /**
         * 步骤 2：使用 for 循环遍历 1 到 20 的奇数
         * Step 2: Use for loop to iterate through odd numbers from 1 to 20
         * i 从 1 开始，每次增加 2，生成奇数序列

         */

            /**
             * 步骤 3：打印当前奇数
             * Step 3: Print the current odd number
             */
            System.out.println("Odd number is: " + i);
        }
    }
}

/**
 * 偶数线程类 - Even Number Thread Class
 * 任务：创建一个线程来打印 1 到 20 之间的偶数
 * Task: Create a thread to print even numbers between 1 and 20
 */
class EvenNumber extends Thread {
    /**
     * 步骤 1：重写 run 方法，定义线程执行的任务
     * Step 1: Override the run method to define the thread's execution task
     */
    @Override
    public void run() {
        /**
         * 步骤 2：使用 for 循环遍历 2 到 20 的偶数
         * Step 2: Use for loop to iterate through even numbers from 2 to 20
         * i 从 2 开始，每次增加 2，生成偶数序列
         * i starts from 2 and increments by 2 each time, generating even
number sequence
         */
                    /**
             * 步骤 3：打印当前偶数
             * Step 3: Print the current even number
             */
            System.out.println("Even number is: " + i);
        }
    }
}

/**
 * 多线程演示类 - Multithreading Demo Class
 * 任务：创建并启动奇数线程和偶数线程
 * Task: Create and start odd number thread and even number thread
 */
public class Demo1 {
    /**
     * 主方法 - Main Method
     * 步骤 1：创建奇数线程对象
     */
    public static void main(String[] args) {
        // 步骤 1：创建奇数线程实例
        // Step 1: Create odd number thread instance

        // 步骤 2：创建偶数线程实例


        // 步骤 3：启动奇数线程（调用 start()方法而不是 run()方法）
        // Step 3: Start the odd number thread (call start() method instead
of run())


        // 步骤 4：启动偶数线程
        // Step 4: Start the even number thread


        /**
         * 注意：由于线程调度是由操作系统控制的，所以奇数偶数的打印顺序可能每次运行都不同
         * Note: Since thread scheduling is controlled by the operating
system,
         * the printing order of odd and even numbers may vary each time you
run the program
         */
    }
}
```

Demo 2:

Creating Thread by implementing Runnable Interface

Program to create two threads to Print Odd number and Even number at the same time by Implementing Runnable Interface.

```java
package chapter11;

// Step 1: Make a class for even numbers that uses Runnable interface
// 第一步：创建一个用于偶数的类，使用 Runnable 接口
class Even1 implements Runnable {
    @Override
    public void run() {
        // Step 2: Make a loop that goes 2,4,6,8...20
        // 第二步：创建一个循环，输出 2,4,6,8...20



        // Step 3: Print each even number
            // 第三步：打印每个偶数
            System.out.println("Even Number is:" + i);
        }
    }
}

// Step 4: Make a class for odd numbers that uses Runnable interface
// 第四步：创建一个用于奇数的类，使用 Runnable 接口
class Odd1 implements Runnable {
    @Override
    public void run() {
        // Step 5: Make a loop that goes 1,3,5,7...19
        // 第五步：创建一个循环，输出 1,3,5,7...19


            // Step 6: Print each odd number
            // 第六步：打印每个奇数
            System.out.println("Odd Number is:" + i);
        }
    }
}

public class Demo2 {
    public static void main(String[] args) {
        // Step 7: Create the even number task object
        // 第七步：创建偶数任务对象



        // Step 8: Create the odd number task object
        // 第八步：创建奇数任务对象
        Odd1 odd1 = new Odd1();

        // Step 9: Make a Thread for even numbers
        // 第九步：为偶数创建一个线程
        // We pass the even1 object to Thread constructor
        // This tells the thread what work to do
        // 我们将 even1 对象传递给 Thread 构造函数
        // 这告诉线程要做什么工作

        // Step 10: Make a Thread for odd numbers
        // 第十步：为奇数创建一个线程
        // We pass the odd1 object to Thread constructor
        // This tells the thread what work to do
        // 我们将 odd1 对象传递给 Thread 构造函数
        // 这告诉线程要做什么工作

        // Step 11: Start the even number thread
        // 第十一步：启动偶数线程


        // Step 12: Start the odd number thread
        // 第十二步：启动奇数线程

    }
}
```

Demo 3:

Creating Multiple thread of the same class.

Program to Print three set of numbers at the same time,1 to 10 for each set.

```
package chapter11;

// Step 1: Make a class that extends Thread
// 第一步：创建一个继承 Thread 的类
class NumberThread1 extends Thread {

    // Step 2: Make a constructor that takes a name
    // 第二步：创建一个接受名称的构造函数
    public NumberThread1(String name) {
        // Step 3: Call parent Thread constructor with the name
        // 第三步：使用名称调用父类 Thread 的构造函数
        super(name);
    }

    @Override
    public void run() {
        // Step 4: Make a loop from 0 to 10
        // 第四步：创建一个从 0 到 10 的循环
        for (int i = 0; i <= 10; i++) {
            try {
                // Step 5: Make the thread sleep for 1000 milliseconds (1
second)

                // 第五步：让线程睡眠 1000 毫秒 (1 秒)
                Thread.sleep(1000);

                // Step 6: Print the thread name and current number
                // 第六步：打印线程名称和当前数字
                System.out.println(Thread.currentThread().getName() + ": " +
i);

            } catch (InterruptedException e) {
                // Step 7: Handle if sleep is interrupted
                // 第七步：处理睡眠被中断的情况
                System.out.println("Sleep interrupted");
            }
        }
    }
}

public class Demo3 {
    public static void main(String[] args) {
        // Step 8: Create first thread with name "Thread1"
        // 第八步：创建名为"Thread1"的第一个线程


        // Step 9: Create second thread with name "Thread2"
        // 第九步：创建名为"Thread2"的第二个线程


        // Step 10: Create third thread with name "Thread3"
        // 第十步：创建名为"Thread3"的第三个线程

        // Step 11: Start the first thread
        // 第十一步：启动第一个线程


        // Step 12: Start the second thread


        // Step 13: Start the third thread
        // 第十三步：启动第三个线程

    }
}
```

Demo 4:

Isalive() and join(0 method:(Modification of the Demo1 Program)

```java
package chapter11;


class OddNumber extends Thread {

    @Override
    public void run() {

        for(int i = 1; i <= 20; i += 2) {

            System.out.println("Odd number is: " + i);
        }
    }
}

class EvenNumber extends Thread {

    @Override
    public void run() {

        for(int i = 2; i <= 20; i += 2) {

            System.out.println("Even number is: " + i);
        }
    }
}

/**
 * 多线程演示类 - Multithreading Demo Class
 * 任务：使用join()方法确保先打印奇数后打印偶数
 * Task: Use join() method to ensure odd numbers print first, then even
numbers
 */
public class Demo1 {

    public static void main(String[] args) {
        // 步骤1：创建奇数线程实例
        // Step 1: Create odd number thread instance
        OddNumber odd = new OddNumber();

        // 步骤2：创建偶数线程实例
        // Step 2: Create even number thread instance
        EvenNumber even = new EvenNumber();

        System.out.println("Before starting threads:");
        System.out.println("Odd thread alive: " + odd.isAlive());
        System.out.println("Even thread alive: " + even.isAlive());


        odd.start();
        // 步骤 5：检查奇数线程启动后的状态
        // Step 5: Check odd thread status after starting
        System.out.println("After starting odd thread:");
        System.out.println("Odd thread alive: " + odd.isAlive());

        // 步骤 6：使用 join() 等待奇数线程完成
        // Step 6: Use join() to wait for odd thread to finish
        try {
            System.out.println("Waiting for odd thread to finish...");
            odd.join(); // 等待奇数线程完成 wait for odd thread to complete
            System.out.println("Odd thread has finished!");
        } catch (InterruptedException e) {
            System.out.println("Main thread was interrupted");
        }

        // 步骤 7：检查奇数线程完成后的状态
        // Step 7: Check odd thread status after completion
        System.out.println("After odd thread finished:");
        System.out.println("Odd thread alive: " + odd.isAlive());

        // 步骤 8：启动偶数线程（此时奇数线程已经完成）
        // Step 8: Start even thread (now odd thread has finished)
        even.start();

        // 步骤 9：检查偶数线程启动后的状态
        // Step 9: Check even thread status after starting
        System.out.println("After starting even thread:");
        System.out.println("Even thread alive: " + even.isAlive());

        // 步骤 10：使用 join() 等待偶数线程完成
        // Step 10: Use join() to wait for even thread to finish
        try {
            System.out.println("Waiting for even thread to finish...");
            even.join(); // 等待偶数线程完成 wait for even thread to complete
            System.out.println("Even thread has finished!");
        } catch (InterruptedException e) {
            System.out.println("Main thread was interrupted");
        }

        // 步骤 11：检查两个线程的最终状态
        // Step 11: Check final status of both threads
        System.out.println("Final status:");
        System.out.println("Odd thread alive: " + odd.isAlive());
        System.out.println("Even thread alive: " + even.isAlive());

        System.out.println("All threads completed!");
    }
}
```

Demo 5:
Thread Priority

- Priority is just a suggestion to OS
- 优先级只是给操作系统的建议
- OS may ignore thread priorities
- 操作系统可能忽略线程优先级
- Don't rely on priority for program logic
- 不要依赖优先级来实现程序逻辑

```java
package chapter11;

/**
 * Simple thread class to demonstrate thread name and priority
 * 简单的线程类，演示线程名称和优先级
 */
class MyTask extends Thread {

    /**
     * Constructor to set thread name
     * 构造函数设置线程名称
     * @param name the name of the thread 线程名称
     */
    public MyTask(String name) {
        // Step 1: Call parent Thread constructor to set name
        // 步骤 1：调用父类 Thread 构造函数设置名称
        super(name);
    }

    /**
     * Run method - thread execution code
     * Run 方法 - 线程执行代码
     */
    public void run() {
        // Step 2: Print thread information
        // 步骤 2：打印线程信息
        System.out.println("Thread Name: " + this.getName());
        System.out.println("Thread Priority: " + this.getPriority());

        // Step 3: Do some work in the thread
        // 步骤 3：在线程中执行一些工作
        for (int i = 1; i <= 3; i++) {
            System.out.println(this.getName() + " counting: " + i);
            try {
                // Step 4: Small delay to see the execution order
                // 步骤 4：小延迟以观察执行顺序
                Thread.sleep(500);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.out.println(this.getName() + " finished\n");
    }
}

/**
 * Demo class to show thread name and priority settings
 * 演示类，展示线程名称和优先级设置
 */
public class ThreadPriorityDemo {
    public static void main(String[] args) {
        // Step 1: Create threads with different names
        // 步骤 1：创建具有不同名称的线程
        MyTask thread1 = new MyTask("High-Priority-Thread");
        MyTask thread2 = new MyTask("Normal-Priority-Thread");
        MyTask thread3 = new MyTask("Low-Priority-Thread");

        // Step 2: Set different priorities for threads
        // 步骤 2：为线程设置不同的优先级
        thread1.setPriority(Thread.MAX_PRIORITY);    // Priority 10 优先
级 10
        thread2.setPriority(Thread.NORM_PRIORITY);   // Priority 5  优先
级 5
        thread3.setPriority(Thread.MIN_PRIORITY);    // Priority 1  优先
级 1

        // Step 3: Start all threads
        // 步骤 3：启动所有线程
        thread1.start();
        thread2.start();
        thread3.start();

        // Step 4: Print main thread information
        // 步骤 4：打印主线程信息
        System.out.println("Main Thread Name: " +
Thread.currentThread().getName());
        System.out.println("Main Thread Priority: " +
Thread.currentThread().getPriority());
    }
}
```

Demo 6: Interthread communication:

Number Box
Imagine two friends sharing one box.
Friend A (Producer) puts a number inside the box.
Friend B (Consumer) checks if the number is even or odd.

They must take turns:
A waits if the box already has a number.
B waits if the box is empty.

- This is like Producer and Consumer threads sharing one resource.

```
/**
 * Inter-thread Communication Example - Even/Odd Number Checker
 * 线程间通信示例 - 奇偶数检查器
 *
 * This program demonstrates producer-consumer pattern where:
 * - Producer creates numbers
 * - Consumer checks whether number is EVEN or ODD
 *
 * 该程序演示了生产者-消费者模式，其中：
 * - 生产者生成数字
 * - 消费者检查数字是偶数还是奇数
 */

class SharedBox {
    int number;                    // The shared data between threads 线程间共享的数据
    boolean hasValue = false;      // Flag: True means box already has a number 标志：true 表示盒子中已有数字

    /**
     * Producer method - puts a number in the shared box
     * 生产者方法 - 将数字放入共享盒子
     *
     * @param value The number to be produced 要生产的数字
     */
    synchronized void produce(int value) {
        // STEP 1: Check if box already has a value (consumer hasn't consumed yet)
        // 步骤 1: 检查盒子是否已有值（消费者尚未消费）
        while (hasValue) {
            try {
                System.out.println("Producer waiting - box is full");
                // STEP 2: Wait for consumer to consume
                // 步骤 2: 等待消费者消费
                wait(); // Release lock and wait for consumer to consume 释放
锁并等待消费者消费
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // STEP 3: Produce the number and put in box
        // 步骤 3: 生产数字并放入盒子
        number = value;
        hasValue = true;
        System.out.println("Produced: " + number);

        // STEP 4: Notify consumer that number is ready
        // 步骤 4: 通知消费者数字已准备好
        notify();
        System.out.println("Producer notified consumer");
    }

    /**
     * Consumer method - takes number from box and checks even/odd
     * 消费者方法 - 从盒子中取出数字并检查奇偶性
     */
    synchronized void consume() {
        // STEP 1: Check if box is empty (producer hasn't produced yet)
        // 步骤 1: 检查盒子是否为空（生产者尚未生产）
        while (!hasValue) {
            try {
                System.out.println("Consumer waiting - box is empty");
                // STEP 2: Wait for producer to produce
                // 步骤 2: 等待生产者生产
                wait(); // Release lock and wait for producer to produce 释放
锁并等待生产者生产
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // STEP 3: Consume the number and perform logical operation
        // 步骤 3: 消费数字并执行逻辑操作
        if (number % 2 == 0)
            System.out.println("Consumed: " + number + " → Even Number");
        else
            System.out.println("Consumed: " + number + " → Odd Number");

        // STEP 4: Mark box as empty and notify producer
        // 步骤 4: 标记盒子为空并通知生产者
        hasValue = false;
        notify();
        System.out.println("Consumer notified producer");
    }
}

/**
 * Producer Thread Class
 * 生产者线程类
 *
 * Responsible for generating numbers and putting them in the shared box
 * 负责生成数字并将其放入共享盒子
 */
class Producer extends Thread {
    SharedBox box;  // Reference to shared box 对共享盒子的引用

    // Constructor to initialize with shared box
    // 构造函数，使用共享盒子初始化
    Producer(SharedBox b) {
        box = b;
    }

    /**
     * Thread execution method - produces 5 numbers
     * 线程执行方法 - 生产 5 个数字
     */
    public void run() {
        // STEP 1: Loop to produce numbers 1 to 5
        // 步骤 1: 循环生产数字 1 到 5
        for (int i = 1; i <= 5; i++) {
            // STEP 2: Call produce method with current number
            // 步骤 2: 使用当前数字调用生产方法
            box.produce(i);
            try {
                // STEP 3: Simulate production time
                // 步骤 3: 模拟生产时间
                Thread.sleep(400);
            }
            catch (InterruptedException e) {}
        }
        System.out.println("Producer finished producing all numbers");
    }
}

/**
 * Consumer Thread Class
 * 消费者线程类
 *
 * Responsible for taking numbers from box and checking even/odd
 * 负责从盒子中取出数字并检查奇偶性
 */
```

```java
class Consumer extends Thread {
    SharedBox box;  // Reference to shared box 对共享盒子的引用

    // Constructor to initialize with shared box
    // 构造函数，使用共享盒子初始化
    Consumer(SharedBox b) {
        box = b;
    }

    /**
     * Thread execution method - consumes 5 numbers
     * 线程执行方法 - 消费 5 个数字
     */
    public void run() {
        // STEP 1: Loop to consume 5 numbers
        // 步骤 1: 循环消费 5 个数字
        for (int i = 1; i <= 5; i++) {
            // STEP 2: Call consume method
            // 步骤 2: 调用消费方法
            box.consume();
            try {
                // STEP 3: Simulate consumption time
                // 步骤 3: 模拟消费时间
                Thread.sleep(400);
            }
            catch (InterruptedException e) {}
        }
        System.out.println("Consumer finished consuming all numbers");
    }
}

/**
 * Main Class - Program Entry Point
 * 主类 - 程序入口点
 */
public class EvenOddInterThread {

    /**
     * Main method - starts producer and consumer threads
     * 主方法 - 启动生产者和消费者线程
     */
    public static void main(String[] args) {
        // STEP 1: Create shared box object (shared resource)
        // 步骤 1: 创建共享盒子对象（共享资源）
        SharedBox box = new SharedBox();

        // STEP 2: Create producer thread with shared box reference
        // 步骤 2: 使用共享盒子引用创建生产者线程
        Producer p = new Producer(box);

        // STEP 3: Create consumer thread with shared box reference
        // 步骤 3: 使用共享盒子引用创建消费者线程
        Consumer c = new Consumer(box);

        // STEP 4: Start producer thread
        // 步骤 4: 启动生产者线程
        p.start();

        // STEP 5: Start consumer thread
        // 步骤 5: 启动消费者线程
        c.start();

        System.out.println("Both producer and consumer threads started");
    }
}
```