Contents

1. Classes and Objects
2. Inheritance
3. Interfaces
4. Exceptions and Error Handling
5. Intro to Concurrency
6. Concurrency in Java
7. Graphics and Animation
8. Applets
Chapter 6: Concurrency in Java

After this chapter you will be able to:

- Create threads
- Incorporate synchronized methods
- Put threads to sleep using wait
- Wake up threads using notify
Creating Threads

• Two ways to create threads:
  - write a class which implements interface Runnable
    
    ```java
    interface Runnable {
        public void run ();
    }
    ```
  - or write a subclass of class Thread
Using Interface Runnable

• Method `run` will be called when a thread is started by calling `Thread.start`

```java
public class MyThread1 implements Runnable {
    private int id;

    public MyThread1 (int id) { this.id = id; }

    public void run () {
        while (true)
            System.out.println("Message from #: " + id);
    }
}
```

• Note: Implementing `Runnable` is the first way to incorporate threads
Creating MyThread1 Objects

- Class `Thread` has a constructor whose argument is of type `Runnable`

```java
class Thread {
    public Thread (Runnable target) {
        ...
    }
    Thread t1 = new Thread(new MyThread1(1));
    Thread t2 = new Thread(new MyThread1(2));
    t1.start();
    t2.start();
```

Output:

```
Message from #1
Message from #2
Message from #1
Message from #2
...
```
Creating A Subclass of Thread

- Subclassing class Thread is the second way to incorporate threads

```java
public class MyThread2 extends Thread {
    private int id;

    public MyThread2 (int id) { this.id = id; }

    public void run () {
        while (true)
            System.out.println("message from thread: "+ id);
    }
}
```
Creating MyThread2 Objects

Thread t1 = new MyThread2(1);
Thread t2 = new MyThread2(2);

t1.start();
t2.start();
Sleeping

- A thread may call the Thread.sleep method to sleep for a specified number of milliseconds

```java
public class MyThread2 extends Thread {
    private int id;

    public MyThread2 (int id) { this.id = id; }

    public void run () {
        while (true) {
            System.out.println("message from thread: " + id);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                System.out.println("snooze was interrupted");
            }
        }
    }
}
```
InterruptedException

• The InterruptedException is thrown when the interrupt method of a Thread is called

• The interrupt method is defined in class Thread

```java
Thread t1 = new MyThread2(1);

t1.start();
...
ti.interrupt(); // interrupt t1's sleep
```
Synchronization

- **Motivation:** to avoid race conditions
- **Example:** a shared counter must properly synchronize increment operation

- **Java solution:** each *object* has an associated monitor
- **Use keyword** *synchronized* in method declaration
- **Guarantee:** only *one* thread may be executing *any* synchronized method of an object
Synchronized Methods

• Use keyword `synchronized` when necessary to avoid race conditions

```java
public class SharedCounter {
    private int counter = 0;

    public synchronized void increment () {
        counter ++;
    }

    public synchronized void decrement () {
        counter --;
    }
}
```

• Only **one** thread may be executing either method at one time
Waiting

- A thread may go to sleep by calling `wait`
- **Example:** A thread needs to wait until the counter is zero

```java
public class SharedCounter {
    private int counter = 0;
    ...
    public void awaitZero () {
        if (counter != 0)
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("wait was interrupted by another thread");
            }
    }
}
```
Notifying

• Another thread may subsequently call `notify/notifyAll` to wake up sleeping threads

• `notify` wakes up **one** of the sleeping threads

• `notifyAll` wakes up **all** sleeping threads

• **Note:** since only one thread may be active inside a `synchronized` method, an awoken thread will not start executing immediately
Revised Increment/Decrement

- Use `notifyAll` when counter becomes zero

```java
public class SharedCounter {
    private int counter = 0;

    public void awaitZero () {...}

    public synchronized void increment () {
        counter ++;
        if (counter ==0)
            notifyAll();
    }
    public synchronized void decrement () {
        counter --;
        if (counter == 0)
            notifyAll();
    }
}
```
Thread Priorities

• Each thread has a priority which may be accessed via `Thread.getPriority` or changed by calling `Thread.setPriority`.

• Threads with higher priorities are scheduled for execution more often.

Priorities are in the range:

- `MIN_PRIORITY` = 1
- `MAX_PRIORITY` = 10

• The garbage collector runs as a low priority thread.
Changing Priority

- Call `Thread.setPriority`

```java
Thread t1 = new MyThread2(1);
t1.setPriority(9);
```

- If new priority is not in the range `MIN_PRIORITY` to `MAX_PRIORITY` then `IllegalArgumentException` is thrown.
Daemons

• Thread.setDaemon sets a thread to be either a user thread or a daemon thread

Thread t1 = new MyThread2(1);

t1.setDaemon(true);  // must be done before // Thread is started

• Initially a thread is a user thread

• The java virtual machine exits when the only threads running are daemon threads
Thread Status

• Can call Thread.isAlive to determine the status of a thread

```java
Thread t1 = new MyThread2(1);

if (!t1.isAlive()) System.out.println("NOT alive");
t1.start();
if (t1.isAlive()) System.out.println("alive");
t1.stop();
if (!t1.isAlive()) System.out.println("NOT alive");
```

• Thread.stop halts the execution of a thread

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT alive</td>
</tr>
<tr>
<td>alive</td>
</tr>
<tr>
<td>NOT alive</td>
</tr>
</tbody>
</table>
Class ThreadGroup

- Can group threads together using class ThreadGroup

- Class ThreadGroup defines methods that facilitate applying operations to all of the threads in the group
It’s Exercise Time