Amirkabir University of Technology
(Tehran Polytechnic)
Computer Engineering Department

Java Threading

Presenters: Radin Shayanfar, Erfan Ghasemi

Supervisor: Dr. Mahmoud Momtazpour

Spring 2021

Outline

Outline

Threading in Java

How to create threads in Java?

Outline

Threading in Java

How to create threads in Java?

Java Virtual Machine

How JVM executes threads?

Outline

Threading in Java

How to create threads in Java?

Java Virtual Machine

How JVM executes threads?

Low-level Mechanisms

Low-level thread communication

Outline

Threading in Java

How to create threads in Java?

Java Virtual Machine

How JVM executes threads?

Low-level Mechanisms

Low-level thread communication

High-level Mechanisms

More advanced features…

Outline

Threading in Java
How to create threads in Java?

Java Virtual Machine
How JVM executes threads?

Low-level Mechanisms
Low-level thread communication

High-level Mechanisms
More advanced features…

Threading in Java

- Thread Creation in Java
- Approach 1: Extending Thread Class
- Approach 2: Implementing Runnable Interface

Thread Creation in Java

There are two ways to create a thread:

- The new class is a subclass of java.lang.Thread.
- The new class implements java.lang.Runnable interface.

run() method must be implemented in the new class.

This method describes the new thread's task.

1 Approach 1: Extending Thread Class

```
class MyThread extends Thread {
      @Override
      public void run() {
         System.out.println("Hello");
         System.out.println("Bye");
      }
}
public class ThreadExample{
    public static void main(String[] args) {
        System.out.println("Hi");
        MyThread t = new MyThread();
        t.start();
        System.out.println("Good luck");
    }
}
```

- An object should be instanced from MyThread class.
- start() method should be called on it.

Approach 2: Implementing Runnable Interface

```
Class MyRunnable implements Runnable{
      @Override
      public void run() {
         System.out.println("Hello");
         System.out.println("Bye");
      }
}

Thread t = new Thread(new MyRunnable());
t.start();
```

- An object should be instanced from MyRunnable class.
- The MyRunnable instance should be passed to Thread class's constructor.
- Calling the start method on created Thread object will start thread execution.

Outline

Threading in Java

How to create threads in Java?

Java Virtual Machine

How JVM executes threads?

Low-level Mechanisms

Low-level thread communication

High-level Mechanisms

More advanced features…

Java Virtual Machine

- Green Threads
- Threading Implementation in JVM

Green Threads

- Green Threads refers to the name of the original thread library for the programming language Java.
- Green threads or virtual threads are threads that are scheduled by a runtime library or virtual machine.
- Green threads emulate multithreaded environments without relying on any native OS abilities.
- They are managed in user space instead of kernel space.

Threading Implementation in JVM

- In Java 1.1, green threads were the only threading model used by the Java Virtual Machine.
- Java versions dropped green threads in favor of native threads.
- Squawk virtual machine uses green threads to minimize the use of native code, and to support migrating.
- Virtual threads is a lightweight user-mode scheduled alternative to standard OS managed threads.

Outline

Threading in Java

How to create threads in Java?

Java Virtual Machine

How JVM executes threads?

Low-level Mechanisms

Low-level thread communication

High-level Mechanisms

More advanced features…

Low-level Mechanisms

- Synchronized Blocks
- Inter-Thread Communication

Synchronized blocks

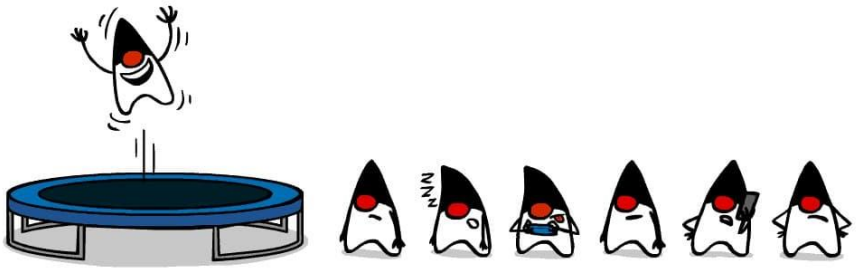
- The synchronized keyword specifies the critical sections.
- There is one lock for each object.
- If an object's synchronized method is called, object's lock should be acquired before entering the method.

```
public class BankAccount{
     private float balance;
     public synchronized void deposit(float amount) {
         balance+= amount;
     }
     public synchronized void withdraw(float amount) {
        balance-= amount;
     }
}
```

3 Synchronized blocks

- It is possible to use other locks to protect critical sections.
- If a static method is defined as synchronized:
  - Each thread should acquire the class's lock instead of object's lock.
  - Therefore no two threads can execute the static method at the same time.

```
List<String> names;
...
synchronized(names){
    names.add("ali");
}
```



Inter-Thread Communication

- Sometimes a thread needs to wait until another thread notifies it.
- These methods are defined in the Object class.
- A thread should have the object's lock to call notify() and wait() methods on that object.
- Thus wait() and notify() of object X should only be called inside synchronized(X) block.
- Each object has a list of waited threads.

```
synchronized (obj) {
  obj.notify();
}
```

Outline

Threading in Java

How to create threads in Java?

Java virtual machine

How JVM executes threads?

Low-level Mechanisms

Low-level thread communication

High-level Mechanisms

More advanced features…

High-level Mechanisms

- Synchronizer Objects
- Semaphore, CountDownLatch, Exchanger, and CyclicBarrier

High-level Threading Mechanisms

- So far, we saw low-level threading mechanisms, which are painful to work with.
- As of Java 5, high-level APIs were added to Java (java.util.concurrent package).
- Easier to use, better performance, and higher utilization on multicore CPUs

Synchronizer Objects

- A shared object is used to synchronize threads.
- Plenty of different classes:
  - Semaphore
  - CountDownLatch
  - Exchanger
  - CyclicBarrier

Semaphore

- Controls access to shared resources.
- Has an internal state to keep the number of allowed threads.
- Primary methods: acquire() and release()
  - acquire() blocks if internal state of semaphore reaches zero.

4 Semaphore: Example

- Two concerns:
  - Threads should not access the list at the same time → `synchronized`
  - Consumer must be blocked if the list is empty → `semaphore`

```
Semaphore sem = new Semaphore(0);
```

```
sem.acquire();
synchronized (list) {
  obj = list.remove(0);
}
```

Consumer

```
synchronized(list){
  list.add(obj);
}
sem.release();
```

Producer

4 CountDownLatch

- Allows multiple threads to wait until certain amount of work is done.
- Major methods:
  - await(): blocks the calling thread until the countdown reaches zero.
  - countdown(): decreases the countdown by one.

```
CountDownLatch latch = new CountDownLatch(2);
```

Thread#1

```
latch.await();
System.out.println("Finished!");
```

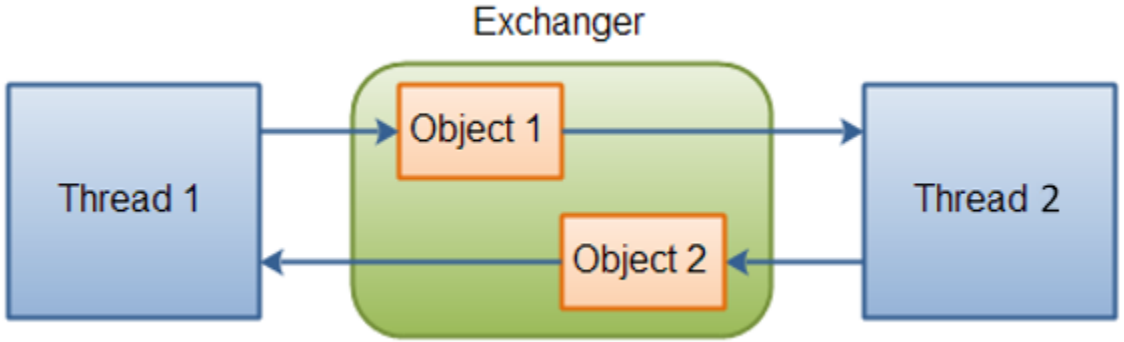
```
latch.countDown();
```

Thread#2

```
latch.countDown();
```

Thread#3

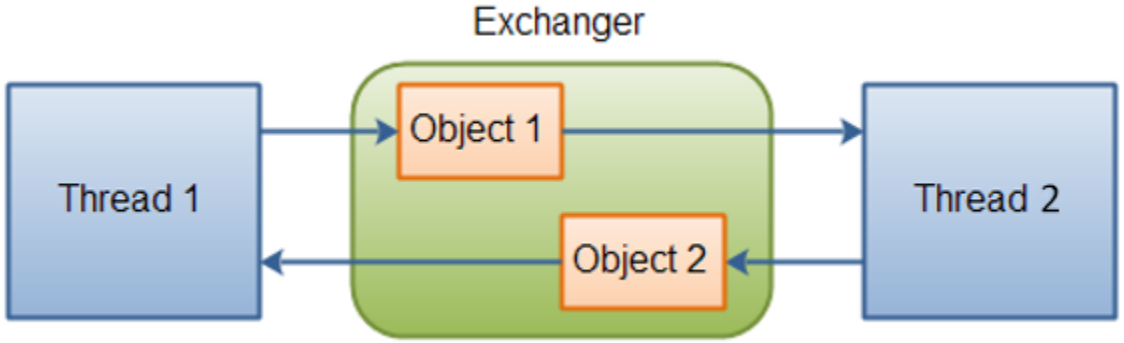
4 More synchronizers



- Exchanger
  - For synchronization and message passing between two threads.

4 More synchronizers

- Exchanger
  - For synchronization and message passing between two threads.

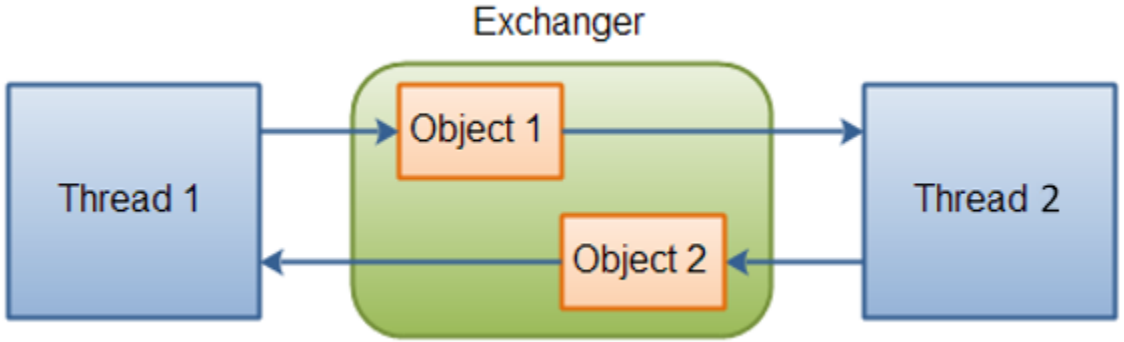



```
Exchanger<String> e = new Exchanger<>();
```

```
e.exchange("x=2");
```
Thread#1

```
e.exchange("y=3");
```
Thread#2

4 More synchronizers



- Exchanger
  - For synchronization and message passing between two threads.

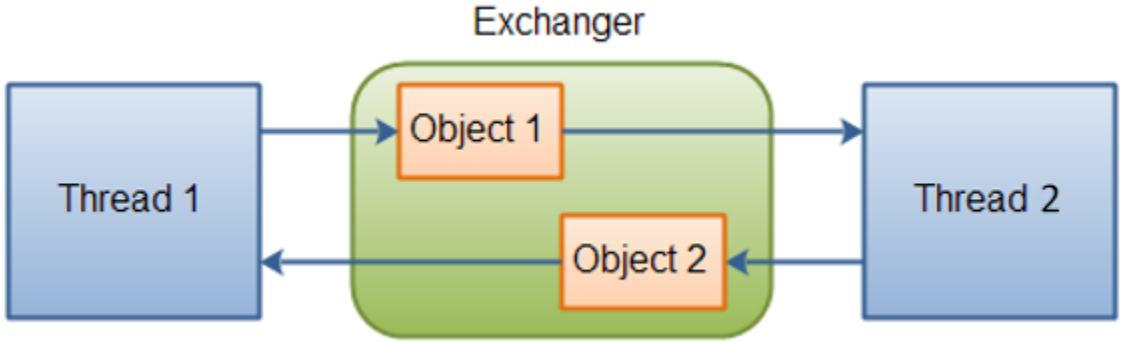
```
Exchanger<String> e = new Exchanger<>();
```

```
e.exchange("x=2");
```
Thread#1

```
e.exchange("y=3");
```
Thread#2

- CyclicBarrier
  - Barrier for specific number of threads

4 More synchronizers



- Exchanger
  - For synchronization and message passing between two threads.

```
Exchanger<String> e = new Exchanger<>();
```

```
e.exchange("x=2");
```
Thread#1

```
e.exchange("y=3");
```
Thread#2

- CyclicBarrier
  - Barrier for specific number of threads

```
CyclicBarrier barrier
      = new CyclicBarrier(3);
```

```
barrier.await();
barrier.await();
barrier.await();
```

Thank You!

Any questions?

Amirkabir University of Technology
(Tehran Polytechnic)
Computer Engineering Department