CS301 Software Development

Instructor: Peter Kemper

Introduction to Multithreading - Part 1

Introduction

- Multithreading allows us to program parallel applications within a single process, a single executing application

- Potential benefits are
  - Speed up
    - On a single-processor architecture (with or without multiple cores)
    - On a multiprocessor architecture
  - Software design
    - For certain types of applications: Multithreading gives better class design
  - UI responsiveness
    - Heavy duty calculation lifted by background threads
    - GUI remains interactive and responsive

- You learn
  - What threads are
  - How to program with threads

Threads

- Thread
  - Program unit that is executed independently
  - Belongs to a particular process
  - Multiple threads of a process run simultaneously
- Virtual machine, runtime environment
  - Executes each thread for some time (e.g. a short time slice)
  - Thread scheduler activates, deactivates threads
  - Creates illusion of threads running in parallel
- Multiprocessor / Multicore architectures
  - Threads can actually run in parallel

Declaring and Running Threads

High level recipe:

- Define class that implements interface Runnable
- Runnable has one method

  void run()
- Place thread action into run method

- Construct object of runnable class
- Construct thread from that object
- Start thread

```
public class MyRunnable
    implements Runnable
{
   public void run()
   {
      // thread action
   }
}
...
Runnable r = new MyRunnable();
Thread t = new Thread(r);
t.start();
```

Life cycle

- Instantiation
- Get going: t.start()
- Be productive: run() is executed
- Termination -> run() terminates

Thread Example

- Run two threads in parallel
  - Each thread prints 10 greetings
  - After each printout, sleep for 100 millisec

```
for (int i = 1; i <= 10; i++)
{
   System.out.println(i + ": " + greeting);
   Thread.sleep(100);
}
```

- All threads should occasionally yield control

  (otherwise called "selfish", could be cumbersome with scheduling)
  - Stimulates scheduler and thread switch
  - Methods
    - yield()
    - sleep(long millis) throws InterruptedException

Thread Example: 2 greeter threads

- Note: output not exactly interleaved

```
1: Hello, World!
1: Goodbye, World!
2: Hello, World!
2: Goodbye, World!
3: Hello, World!
3: Goodbye, World!
4: Hello, World!
4: Goodbye, World!
5: Hello, World!
5: Goodbye, World!
6: Hello, World!
6: Goodbye, World!
7: Hello, World!
7: Goodbye, World!
8: Goodbye, World!
8: Hello, World!
9: Goodbye, World!
```

Thread 1 says hello
Thread 2 says goodbye

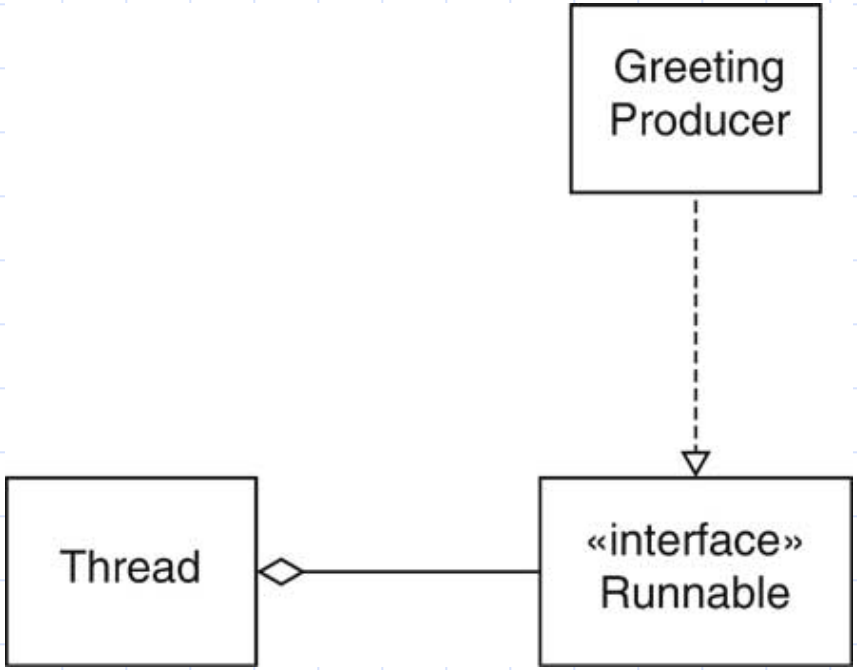


Figure: Cay Horstmann, Object Oriented Design & Patterns, 2nd edition, Wiley

Starting Two Threads

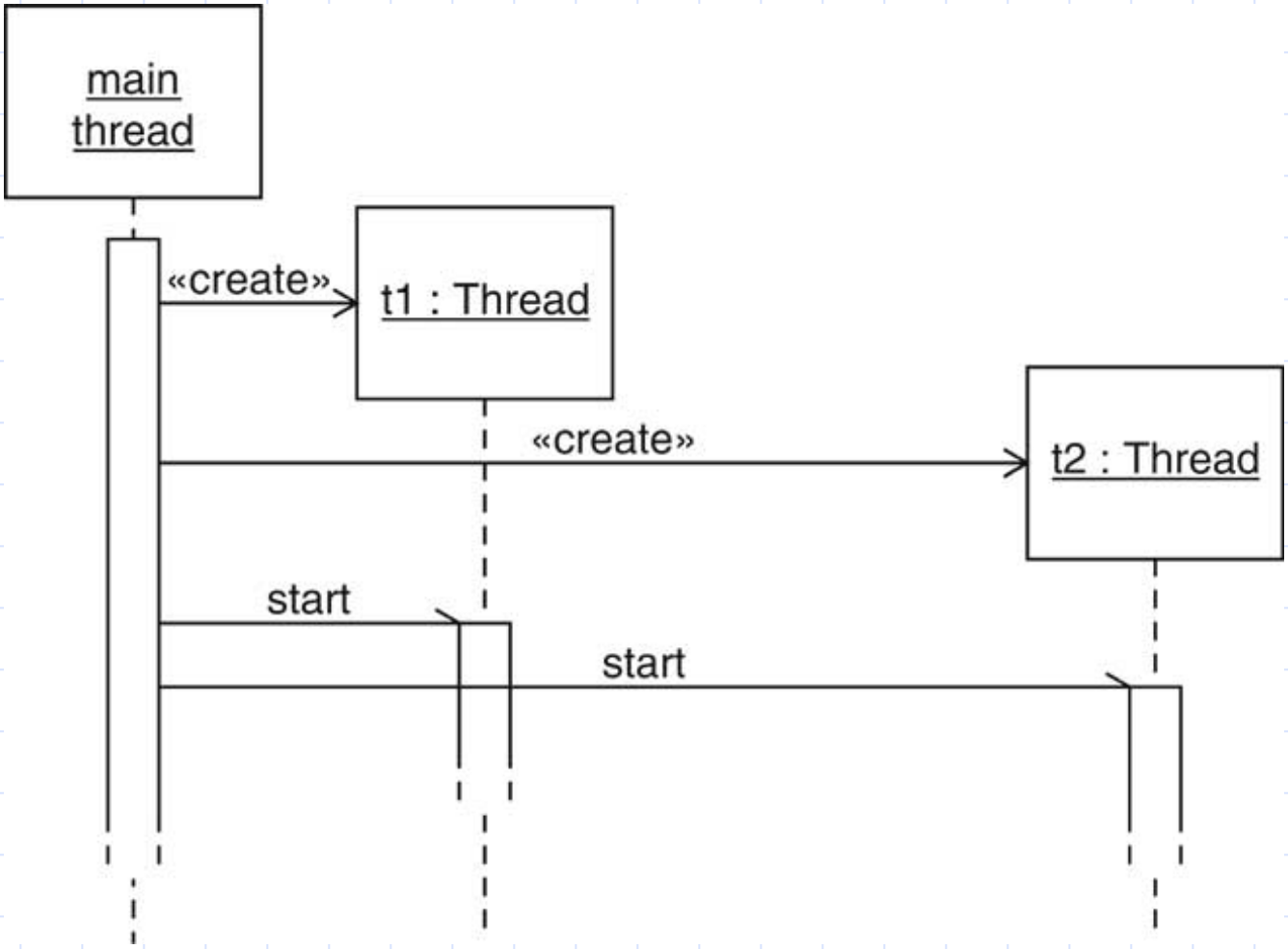
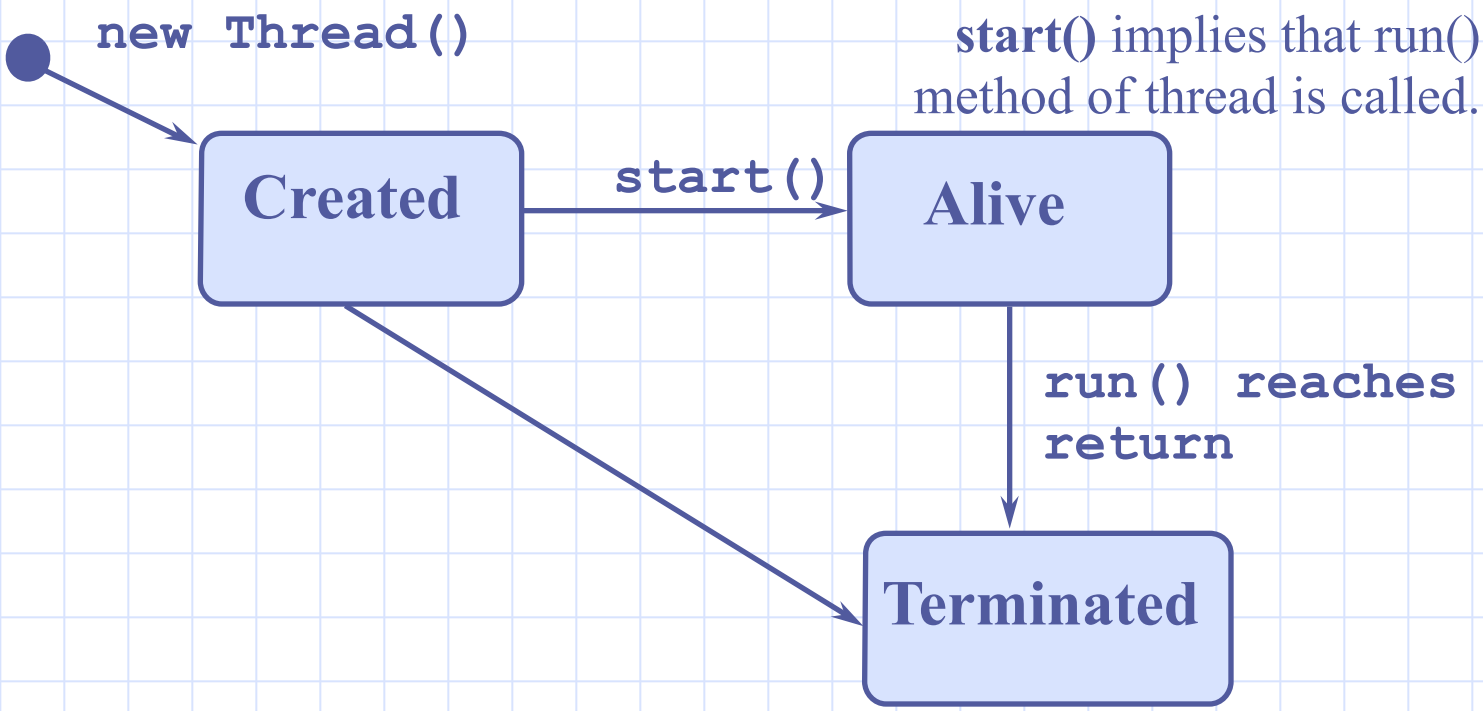


Figure: Cay Horstmann, Object Oriented Design & Patterns, 2nd edition, Wiley

(Simple) Life Cycle of a Thread in Java

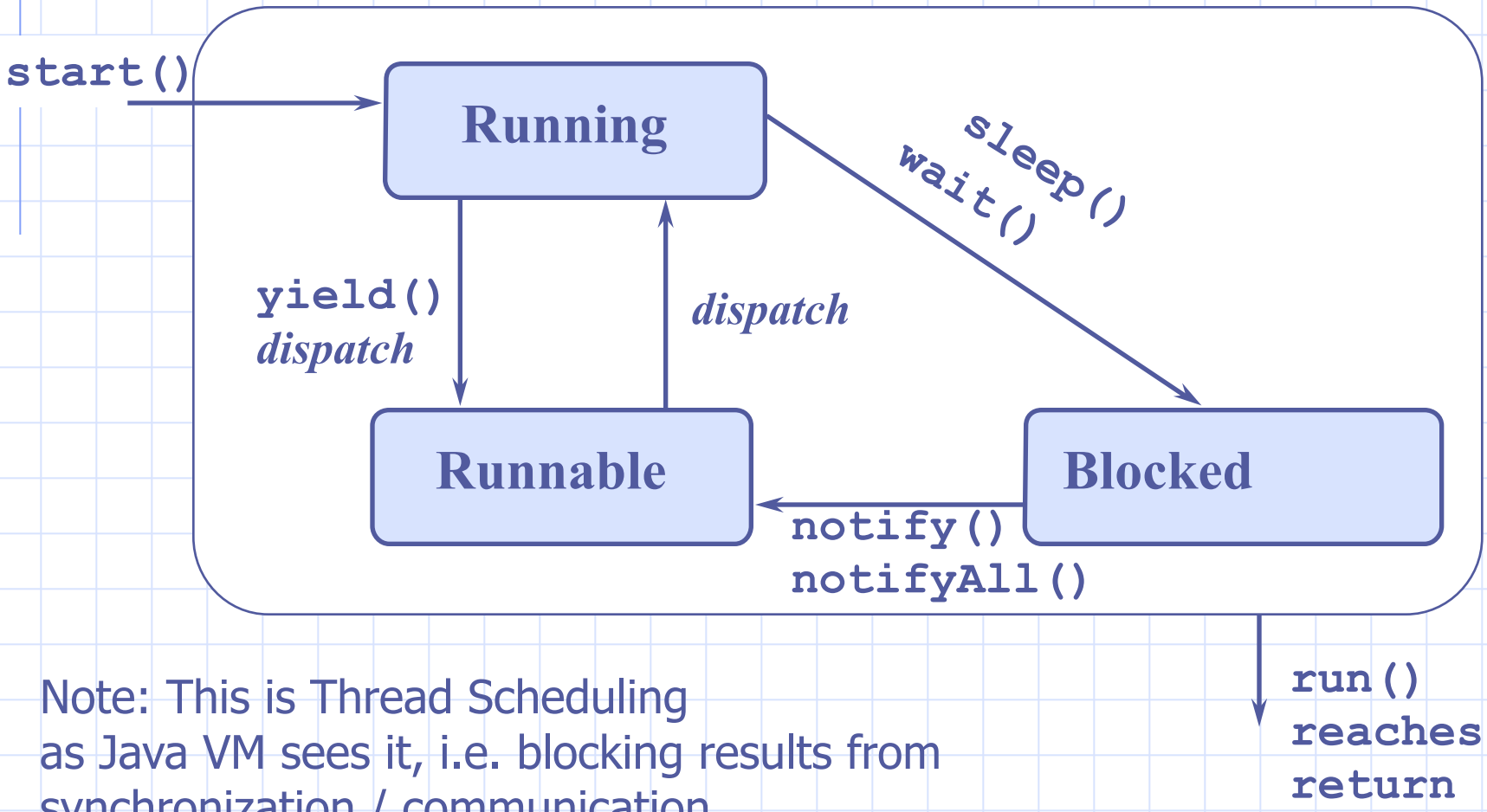
State transition diagram to illustrate life cycle



isAlive() is true,
Once a thread has started and is not terminated yet.
Once terminated, always terminated. (no restart)

States of a Thread that isAlive()

Different states between start() and termination :



Note: This is Thread Scheduling
as Java VM sees it, i.e. blocking results from synchronization / communication.
Underneath: state transitions for OS thread/process scheduling

Blocked Thread State

- Reasons for blocked state:
  - Sleeping
  - Waiting to acquire lock (later)
  - Waiting for condition (later)
- Unblocks only if reason for block goes away

Scheduling Threads

- Scheduler activates new thread if
  - a thread has completed its time slice
  - a thread has blocked itself
  - a thread with higher priority has become runnable
- Scheduler determines new thread to run
  - looks only at runnable threads
  - picks one with max priority

    (for now: don't mess with priorities, platform dependent)

Terminating Threads

A natural death:

- Thread terminates when run exits
- Other thread may wait for it:
  - dyingThread.join()
  - while (dyingThread.isAlive() { Thread.sleep(1000); }

flickr, ©cc from TexasEagle

Terminating Threads

Ways to kill a thread?

1. Murder: Premature approach of early Java versions
   - Stop method: (deprecated)
     - Do NOT use this method,
     - leaves computation in state that is uncontrolled / unclear

flickr, ©cc from Neil Dorgan
Pineapple Psycho

Terminating Threads

Ways to kill a thread?

2. Request for suicide:

   Interrupt method

   - Calling t.interrupt() doesn't actually interrupt t; just sets a flag
   - Effect depends on thread program code!
   - Supposed behavior:

     Interrupted thread must sense interruption

     boolean isInterrupted()

     and exit its run method
   - Interrupted thread has chance to clean up

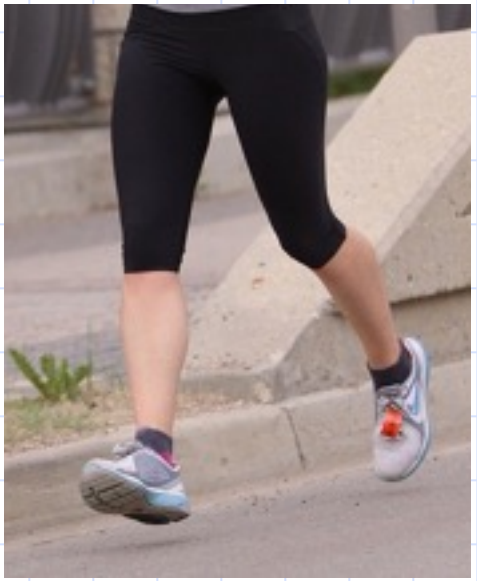
flickr, ©cc from Eddie
Samurai Sword Set

Sensing Interruptions

- Variant 1:
  - Thread could occasionally call
    Thread.currentThread().isInterrupted()
- Variant 2:
  - Thread could occasionally sleep(), or wait()
  - Both methods throw InterruptedException when thread interrupted
  - . . . and then the interruption status is cleared!
- Variant 2 is more robust:
  - Sleep occasionally, catch exception and react to interruption
  - Allows for clear separation of code
- Recommendation:
  - Terminate run when sensing interruption (common reaction)
  - Use variant 2,
    locate all code for clearing up in catch/finally clause
    enhance with variant 1 for lengthy methods without exceptions,
    make check with isInterrupted throw InterruptedException

Sensing Interruptions

```
public class MyRunnable implements Runnable
{
   public void run()
   {
      try
      {
         while (...)
         {
            // do work
            Thread.sleep(...);
         }
      }
      catch (InterruptedException e) {}
      // clean up
   }
}
```



flickr, ©cc from Sangudo
2011 Wild Roses Half Marathon

Sensing Interruptions

```
public class MyRunnable implements Runnable
{
   public void run()
   {
      try
      {
         while (...)
         {
            // do work
            Thread.sleep(...);
         }
      }
      catch (InterruptedException e) {}
      // clean up
   }
}
```

flickr, ©cc from Melissa Sleeping

Sensing Interruptions

```
public class MyRunnable implements Runnable
{
   public void run()
   {
      try
      {
         while (...)
         {
            // do work
            Thread.sleep(...);
         }
      }
      catch (InterruptedException e) {}
      // clean up
   }
}
```

flickr, ©cc from Alan Cleaver
Alarm Clock2

Summary

- Create threads
- Assign work to a thread
- Terminate a thread
- Communicate data between threads
  - Race condition
  - Locks and synchronized method calls