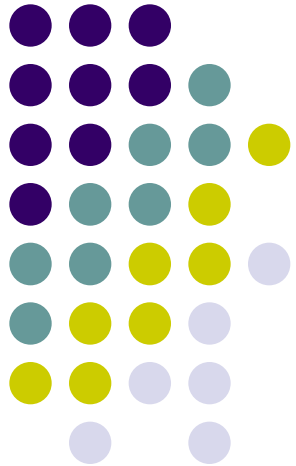
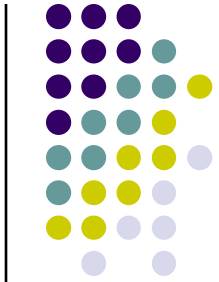
THREADS



By Pakita Shamoi, fall 2017

Plan

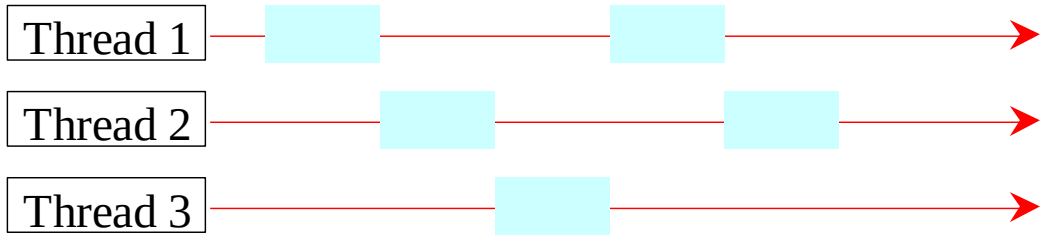


- Threads Concept
- Creating Threads by Extending the `Thread` class
- Creating Threads by Implementing the `Runnable` Interface
- Controlling Threads and Thread Status
- Thread Groups
- Synchronization

Threads Concept

Multiple threads on multiple CPUs

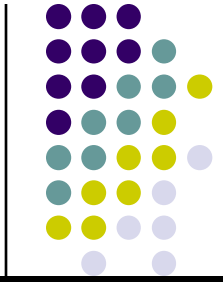
Multiple threads sharing a single CPU

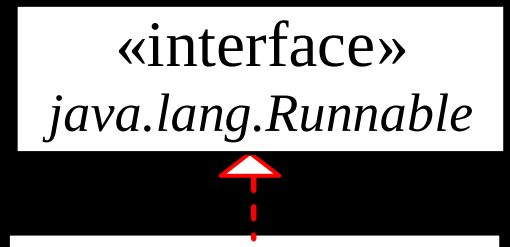


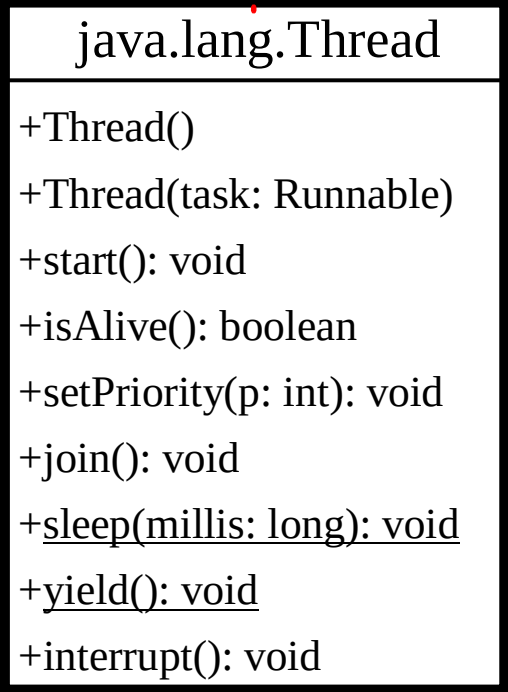
Once `start` method is invoked (which calls the run method), the thread becomes runnable.

Performs time-slicing. Interrupts the running thread periodically to give other threads a chance to run.

The Thread Class







Creates a default thread.

Creates a thread for a specified task.

Starts the thread that causes the run() method to be invoked by the JVM.

Tests whether the thread is currently running.

Sets priority p (ranging from 1 to 10) for this thread.

Waits for this thread to finish.

Puts the runnable object to sleep for a specified time in milliseconds.

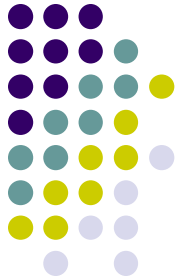
Causes this thread to temporarily pause and allow other threads to execute.

Interrupts this thread.

Two ways to define threads in Java

- Extending the `Thread` class
- Implementing the `Runnable` interface

Creating threads by extending the `Thread` class



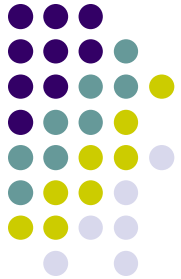
- Define a class, e.g. `NewThread`, extending the `Thread` class
- Override the `run()` method to tell the system how the thread will be executed when it runs.
- Create an instance of `NewThread`,
- Invoke the `start()` method to tell the system to start the thread and to execute the `run()` method.

Creating Threads by Extending the `Thread` class

```
// Custom thread class
public class CustomThread extends Thread
{
  ...
  public CustomThread(...)
  {
    ...
  }

  // Override the run method in Thread
  public void run()
  {
    // Tell system how to run custom thread
    ...
  }

  ...
}
```

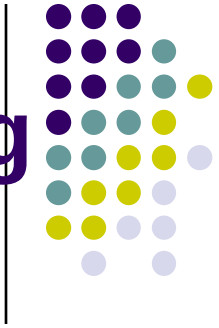
```
// Client class
public class Client
{
  ...
  public someMethod()
  {
    ...
    // Create a thread
    CustomThread thread = new CustomThread(...);

    // Start a thread
    thread.start();
    ...
  }

  ...
}
```

Creating threads by implementing the `Runnable` interface



- Define a class, e.g. `NewTask`, implementing the `Runnable` interface
- implement the `run()` method to tell the system how the task will be executed when it runs.
- Create an instance of `NewTask, e.g. t1`
- The task needs to be executed in a thread. Create an instance of Thread with `t1` as the parameter
- Invoke the `start()` method of the thread to tell the system to start the thread.

Creating Threads by Implementing the `Runnable` Interface



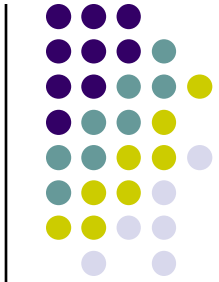
```
// Custom thread class
public class CustomThread
  implements Runnable
{
  ...
  public CustomThread(...)
  {
    ...
  }

  // Implement the run method in Runnable
  public void run()
  {
    // Tell system how to run custom thread
    ...
  }

  ...
}
```

```
// Client class
public class Client
{
  ...
  public someMethod()
  {
    ...
    // Create an instance of CustomThread
    CustomThread customThread
      = new CustomThread(...);

    // Create a thread
    Thread thread = new Thread(customThread);

    // Start a thread
    thread.start();
    ...
  }

  ...
```

Implementing Runnable or Extending Thread

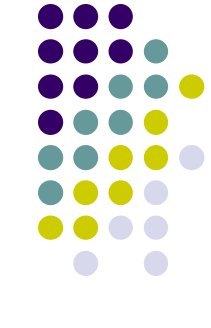
- Use `Runnable` if need to extend another class
- Implementing the Runnable interface is the preferred method for creating a thread if we only want to define the run() method, and not using the other methods in Thread.

At home:



- Create and run three threads in 2 different ways:
  - The first thread prints the letter *a* 100 times.
  - The second thread prints the letter *b* 100 times.
  - The third thread prints the integers 1 through 100.

Controlling Threads and Thread States



- **`void run()`**

  Invoked by the Java runtime system to execute the thread. You must override this method and provide the code you want your thread to execute.

- **`void start()`**

  Starts the thread, which causes the run() method to be invoked. Called by the runnable object in the client class.

- **`static void sleep`**`(long millis) throws InterruptedException`

  Puts the runnable object to sleep for a specified time in milliseconds.

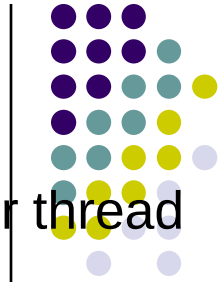
The Static `sleep` method

The sleep(long mills) method puts the thread to sleep for the specified time in milliseconds.

```
public void run() {
  for (int i = 1; i <= lastNum; i++) {
    System.out.print(" " + i);
    try {
      if (i >= 50) Thread.sleep(1000);
    }
    catch (InterruptedException ex) {
    }
  }
}
```

Every time a number (>= 50) is printed, the thread is put to sleep for 1 millisecond.

The join() Method

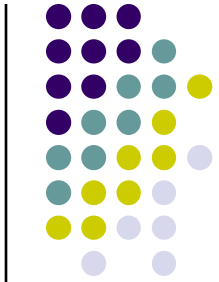


You can use the join() method to force one thread to wait for another thread to finish.

```
public void run(){
Thread thread4 = new Thread(new PrintChar('c', 60));
thread4.start();
try {
   for (int i = 1; i < lastNum; i++){
         System.out.print(" " + i);
         if (i == 50) thread4.join();
       }
   }
   catch (InterruptedException ex) {
       }
}
```

The numbers after 50 are printed after thread4 is finished.

Controlling Threads and Thread States, cont.



- **`void stop`**`()`

  Stops the thread.

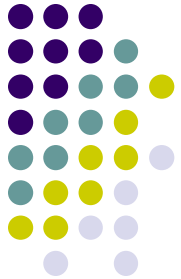
- **`void suspend`**`()`

  Suspends the thread. Use the `resume()` method to resume.

- **`void resume()`**

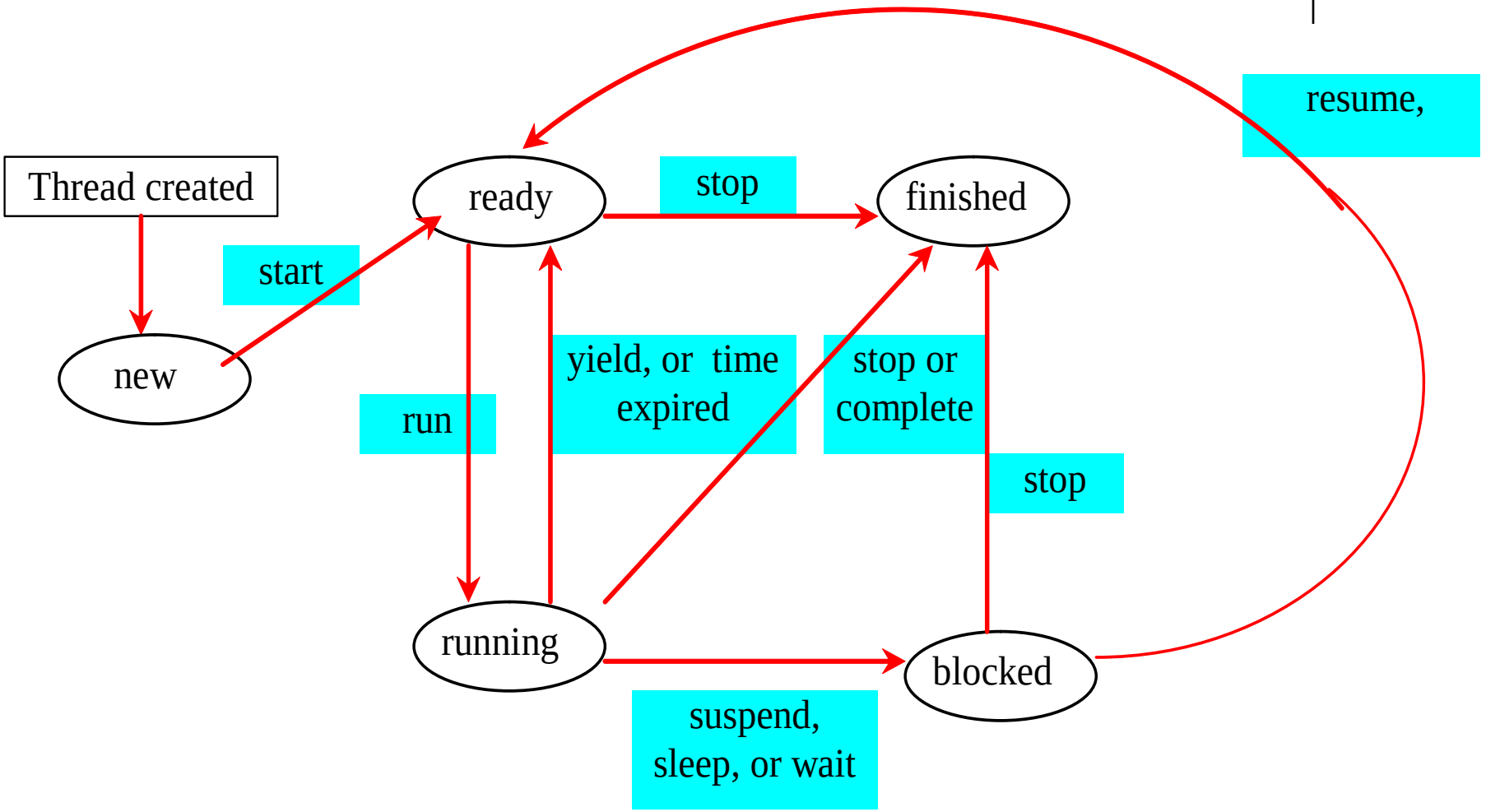
  Resumes the thread suspended with the `suspend()` method.

Thread Priority

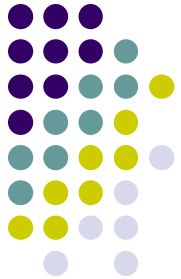


- Each thread is assigned a default priority of `Thread.NORM_PRIORITY.`
- You can reset the priority using `setPriority(int priority).`
- Some constants for priorities include `Thread.MIN_PRIORITY`, `Thread.MAX_PRIORITY`, `Thread.NORM_PRIORITY`

Thread States



Thread Groups



- Construct a thread group using the **`ThreadGroup`** constructor:

```
ThreadGroup g = new ThreadGroup("timer thread group");
```

- Place a thread in a thread group using the `Thread` constructor:

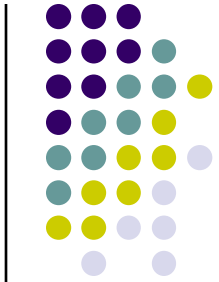
```
Thread t = new Thread(g, new ThreadClass(), "This thread");
```

Thread Groups, cont.

- To find out how many threads in a group are currently running, use the **`activeCount()`** method:

```
System.out.println("The number of “
  + “ runnable threads in the group ” +
  g.activeCount());
```

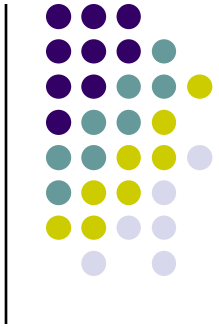
Synchronization



A shared resource may be corrupted if it is accessed simultaneously by multiple threads. For example, two unsynchronized threads accessing the same bank account causes conflict.

Step	balance	thread[i]	thread[j]
1	0	newBalance = bank.getBalance() + 1;	
2	0		newBalance = bank.getBalance() + 1;
3	1	bank.setBalance(newBalance);	
4	1		bank.setBalance(newBalance);

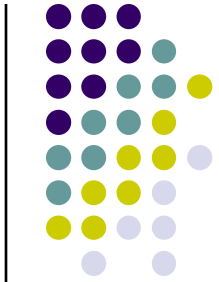
The `synchronized` keyword



It is necessary to prevent more than one thread from simultaneously entering certain part of the program, known as critical region. To avoid resource conflicts, Java uses the keyword synchronized to synchronize method invocation so that only one thread can be in a method at a time.

```
public synchronized void deposit(double amount)
```

Synchronizing Statements vs. Methods



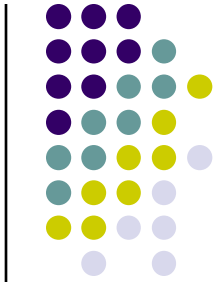
Any synchronized instance method can be converted into a synchronized statement. Suppose that the following is a synchronized instance method:

```
public synchronized void xMethod() {
  // method body
}
```

This method is equivalent to

```
public void xMethod() {
  synchronized (this) {
    // method body
  }
}
```

Review



- You can create threads in Java either by extending Thread or implementing Runnable

- Keep in mind that a shared resource may be corrupted if it is accessed simultaneously by multiple threads