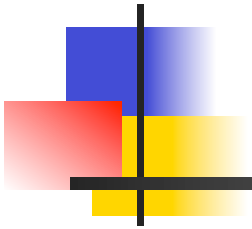


COMP 211: Review of Parallel Programming Concepts (Lectures 31, 34, 35, 39)

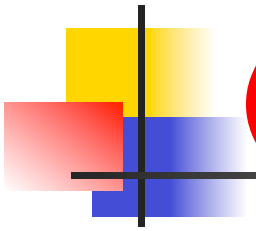
Vivek Sarkar
Department of Computer Science
Rice University

Anonymous Inner classes in Java (Lecture 31, slide 9)

```
public void start(final double rate)
{
    ActionListener adder = new
            ActionListener()
            { // anonymous inner class that implements ActionListener interface
                public void actionPerformed(ActionEvent evt)
                {
                        double interest = balance * rate / 100;
                        balance += interest;
                }
            };
    Timer t = new Timer(1000, adder);
    t.start();
     . . .
}
```

- This is saying, construct a new object of a class that implements the ActionListener interface, where the one required method (actionPerformed) is defined inside the brackets.

Java's Callable Interface (Lecture 31, slide 14)



- Introduced in J2SE 5.0 in java.util.concurrent package (remember to "import java.util.concurrent;")

```
public interface Callable<V> {
      /**
       * Computes a result, or throws an exception.
       *
       * @return computed result
       * @throws Exception if unable to compute a result
       */
      V call( ) throws Exception;
}
```

Task Decomposition using Callable (Lecture 31, slide 15)

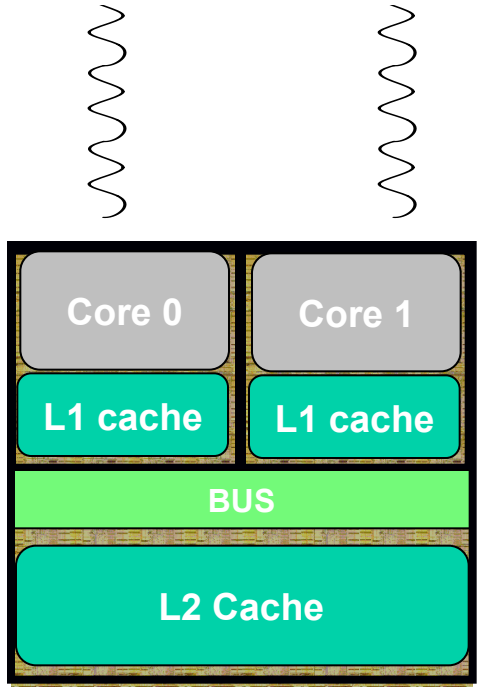
```
// HTML renderer before decomposition
   ImageData image1 = imageInfo.downloadImage(1);
   ImageData image2 = imageInfo.downloadImage(2);
   . . .
   renderImage(image1);
   renderImage(image2);

// HTML renderer after task decomposition
   Callable<ImageData> task1 = new Callable<ImageData>() {
     public ImageData call() {return imageInfo.downloadImage(1);}};
   Callable<ImageData> task2 = new Callable<ImageData>() {
     public ImageData call() {return imageInfo.downloadImage(2);}};
   . . .
   renderImage(task1.call());
   renderImage(task2.call());
```

From Sequential to Parallel Task Decomposition (Lecture 34, slide 18)

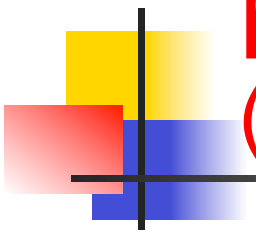
Key Observation:

If two *functional* tasks can be executed in any order, they can also be executed *in parallel*



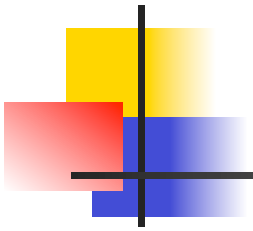
Schematic of a Dual-core Processor

Executing a Callable task in a parallel Java Thread (Lecture 34, slide 20)



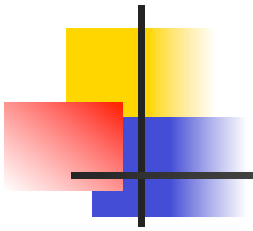
```
// 1. Create a callable closure (lambda)
Callable<ArrayList<Integer>> left_c = …
// 2. Package the closure as a task
final FutureTask<ArrayList<Integer>> task_A =
    new FutureTask<ArrayList<Integer>>(left_c);
// 3. Start executing the task in a parallel thread
new Thread(task_A).start();
// 4. Wait for task to complete, and get its result
left_s = task_A.get();
```

Quicksort with Parallel Tasks (Lectures 34 & 39)



```
public static ArrayList<Integer> quickSort(ArrayList<Integer> a) {
    if (a.isEmpty()) return new ArrayList<Integer>();
    final ArrayList<Integer> left = new ArrayList<Integer>();
    final ArrayList<Integer> mid = new ArrayList<Integer>();
    final ArrayList<Integer> right = new ArrayList<Integer>();
    int pivot = a.get(a.size()/2); // Use midpoint element as pivot
    for (Integer i : a)
        if ( i < pivot ) left.add(i); // Use element 0 as pivot
        else if ( i > pivot) right.add(i);
        else mid.add(i)
    // Now, left, mid, right contain the three partitions of
    // array a with respect to pivot
    // Continue on next slide ...
```

Quicksort with Parallel Tasks (contd) (Lectures 34 & 39)

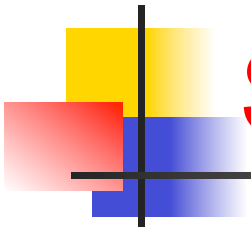


```
  FutureTask<ArrayList<Integer>> left_t = // Closure for recursive call
    new FutureTask<ArrayList<Integer>>(
      new Callable<ArrayList<Integer>>() {
        public ArrayList<Integer> call() { return quickSort(left); } } );
  FutureTask<ArrayList<Integer>> right_t = // Closure for recursive call
    new FutureTask<ArrayList<Integer>>(
      new Callable<ArrayList<Integer>>() {
        public ArrayList<Integer> call() { return quickSort(right); } } );
  // Execute each closure in a parallel thread
  new Thread(left_t).start(); new Thread(right_t).start();
  // Wait for result of FutureTask's left_t and right_t
  ArrayList<Integer> left_s = left_t.get(); // Sorted version of left
  ArrayList<Integer> right_s = right_t.get(); // Sorted version of right
  boolean b = left_s.addAll(mid); b = left_s.addAll(right_s); return left_s;
} // quickSort
```

Example of long-running task with user feedback in GUI (Lecture 35, slide 22)

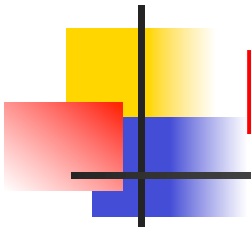
```
private void longRunningTaskWithFeedback() {
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button.setEnabled(false);
       label.setText("busy");
       new Thread(new Runnable() {
         public void run() {
           try { /* Do big computation */ }
           finally {
             GuiExecutor.instance().execute(
               new Runnable() {
                 public void run() {
                   button.setEnabled(true);
                   label.setText("idle");
                 }}); // Runnable for GuiExecutor
           }}}).start(); // Runnable for new Thread
    }}); // ActionListener
} // Source: http://www.javaconcurrencyinpractice.com/listings/ListenerExamples.java
```

Note: `GuiExecutor` is available as open source code from the above site, but is NOT part of the Java Concurrent utilities library.



Summary of Lecture 39

- Trade-offs in Parallel Programming
  - Overhead
  - Memory
  - Serialization
- Computation Graph, Total Work (T_1), Critical Path Length (T_∞)
- Lower bounds in Computation Graph
  - $T_P \geq T_1/P$
  - $T_P \geq T_\infty$
- Amdahl's Law (for serial fraction, f_S, and parallel fraction, f_P)
  - $T_P >= f_S * T_1 + f_P * T_1 / P$



Life beyond COMP 211

- Computer Science & Engineering has a lot to offer
  - Help solve major challenges facing the world
    - Energy, health care, national security, urban planning, …
  - Work on intellectually stimulating problems
    - Cancer prevention, climate modeling, cloud computing, computational science, data mining, dynamics of social/financial networks, mobile computing, …
  - Vast choice of career options
    - Animation, Design, Finance, Information Technology (hardware/ software/services), Government, Law, Medicine, …
- Talk to your major advisor!