ITCS 4145/5145 Assignment 1
Using the Seeds Pattern Programming Framework
1 - Workpool Pattern

Author: B. Wilkinson. Modification date: August 29, 2012

The purpose of this assignment is to become familiar with the Seeds pattern programming framework.[1] This framework was developed at UNC-Charlotte by Jeremy Villalobos as part of his PhD research into pattern programming. The framework is Java-based. The programmer first selects a particular pattern for his application - various patterns are available including workpool, pipeline and synchronous stencil, and others can be created. We shall use the workpool pattern in this assignment as it is very general and applicable to many problems. It also dynamically load balances. In the workpool pattern, a master node sends out data to slave workers. The slaves perform computations and return results to the master, which then produces the final results. To do this, the programmer must implement a Java interface with three principal methods:

- The diffuse method – used by the master to distribute pieces of the data to the slaves.
- The compute method – used by the slaves for the actual computation
- The gather method – used by the master to gather the results

An additional "data count" method is used to tell the framework how many pieces of data will be computed. The programmer might implement a few other methods depending upon the application, notably an initialization method and a method to compute the final result. No message passing routines are needed by the programmer - the diffuse method will create an object with data to be passed to the slaves, and similarly the compute method will create an object with data to the passed back to the master (using generic typing). The framework takes care of the message passing and in fact self deploys on a local computer, a cluster, or a geographically distributed Grid platform when the application is launched. A text file called Availableservers.txt is used to specific the computers. Deployment is done using a second "bootstrapping" class with a main method. This class is mostly written for each pattern and the programmer simply fills in site-specific details (paths, file names, etc.) prior to running.

In this short assignment, we will begin with pre-written code to compute π using the Monte Carlo approach and deploy the framework on a single PC. The Eclipse IDE will be used to provide tools for coding errors. The assignment can be done on your own computer or a lab computer. The main purpose of this assignment is to become familiar with the Seeds approach.

Acknowledgment. This assignment is based upon tutorials written by Jeremy Villalobos, see http://coit-grid01.uncc.edu/seeds/

Note: Seeds is still a prototype with somewhat limited documentation. Anyone interested in improving the project are most welcome. See B. Wilkinson.

[1] Originally the framework was called Parallel Grid Application Framework - "Seeds" comes from the mechanism of "seeding" computers with the framework folder during self-deployment.

Task 1: Install Eclipse

If you do not already have Eclipse installed (generally any version will work), obtain it from http://www.eclipse.org. Download the version of "Eclipse IDE for Java Developers" suitable for your platform. There is no installation wizard so once you have downloaded the compressed Eclipse file, uncompress it and place the Eclipse folder in a suitable place in your file system, typically in C:\Program Files on a Windows XP/7 machine or /usr/localin/ on Linux. The following instructions assume a Windows system.

You also need Java JDK. If you do not already have Java JDK installed, obtain it from http://www.oracle.com/technetwork/java/javase/downloads/index.html. The assignment has been tested with JDK 1.6.0. To determine whether you have Java on a Windows system and if so, its version, type java –version at the command line. A Windows command line can be started by going to Start > Run and typing cmd (Windows XP) or Start and search for "command line" (Windows 7).

Eclipse is started by double clicking the Eclipse executable found in the Eclipse folder. It is convenient to create a shortcut of the Eclipse executable on the desktop. Starting the Eclipse executable will generally ask if you want to use the default location for the workspace, where all your code will be placed. You can select the default location, C:\Documents and Settings\Administrator\workspace.

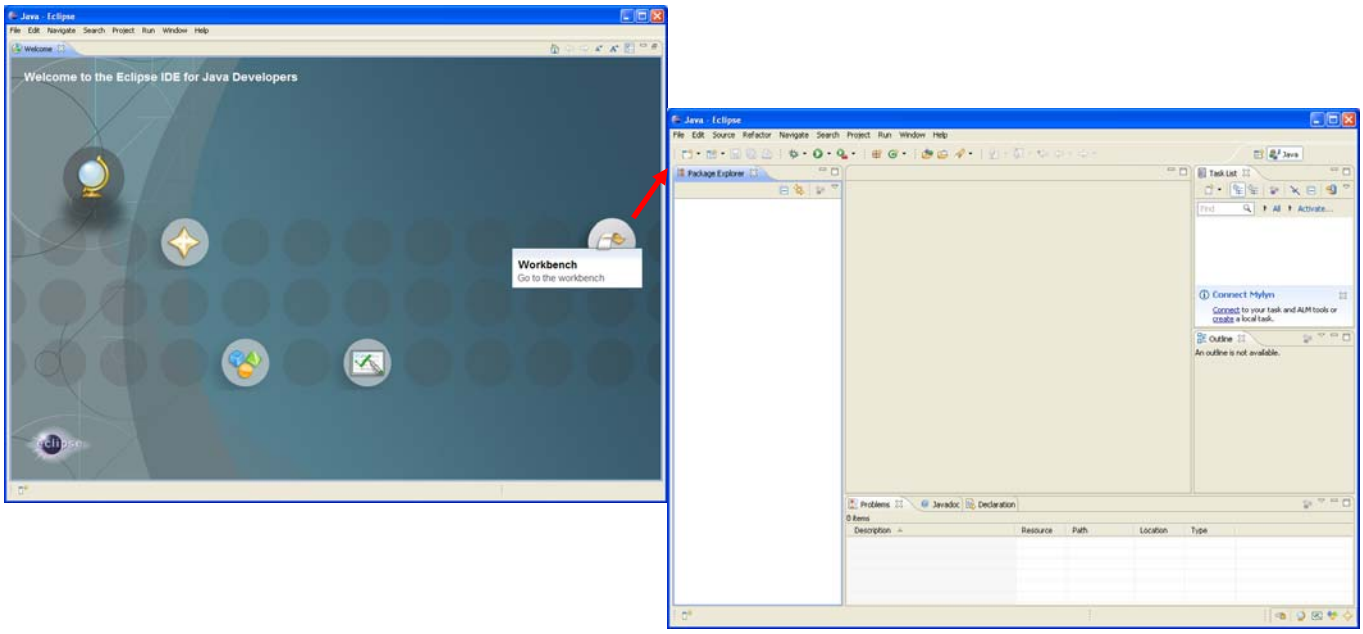


Figure 1 Welcome screen and workbench for Eclipse IDE for Java Developers

Task 2: Install Seeds

Obtain the Seeds framework from the link "Seeds 2.0" under Assignment 1 on the course home page.[2] There is no installation wizard so once you have downloaded the compressed file, uncompress it and place the seeds_2.0 folder in a suitable place in your file system, typically in C:\Program Files on a Windows XP machine.

The "AvailableServers.txt" file found inside the pgaf folder needs to hold information on each computer being used, see the contents of the sample file provided. For this assignment, we will only use a local computer and just need to provide its name and number of cores (actually number of nodes for our workpool). Modify the one uncommented line:

Kronos local - - - 1 11 GridTwo

replacing Kronos with the name of your computer and set the number of cores to 5 (from 11 above). The name of your computer can be found by typing hostname on a Windows command line. Lines starting with a # are commented out lines. Seeds uses port 50040 as default.

Task 3: Monte Carlo π Code

The Monte Carlo algorithm for computing π is well known and given in many parallel programming texts including the course textbook (Wilkinson and Allen 2005]. It is a so-called embarrassingly parallel application particularly amenable to parallel implementation but used more for demonstration purposes than as a good way to compute π. (It can lead to more important Monte Carlo applications.) A circle is formed within a square. The circle has unit radius and the square has sides 2 x 2. The ratio of the area of the circle to the area of the square is given by $\pi(1^2)/(2 \times 2) = \pi/4$. Points within the square are chosen randomly and a score is kept of how many points happen to lie within the circle. The fraction of points within the circle will be $\pi/4$, given a sufficient number of randomly selected samples. Implementing the pattern in the framework requires two classes. A module Java class implements a pattern interface. It is used to define the flow of information through the pattern. The second class is the boot-strapping class. It is used to define the environment where the application will run.

In the workpool approach, the master process will send a different random number to each of the slaves. Each slave uses that number as the starting seed for their random number generator. The Java Random class nextDouble method returns a number uniformly distributed between 0 and 1.0 (excluding 0 and 1). Each slave then gets the next two random numbers as the coordinates of a point (x,y) using nextDouble. If the point is within the circle (i.e. $x^2 + y^2 < 1$), it increments a counter that is counting the number of points within the circle. This is repeated for 1000 points. Each slave returns its accumulated count. The gatherData method performed by the master accumulates the slave results. A separate method, getPi, executed within the bootstrap module, computes the final approximation for π using the accumulated total.

[2] Or from http://coit-grid01.uncc.edu/seeds/download.php. Select "PGAF Shuttle Folder (Bin)" Seeds_Bin.2.0.zip file (for Windows, .tar file for Linux).

Download Program. Download the sample source code for the Monte Carlo π workpool from the link "Pi Approx" under Assignment 1 on the course home page. Once you have downloaded the compressed file, uncompress it and place the PiApprox folder in a convenient place in your file system (desktop for now). The directory structure is \PiApprox\src\edu\uncc\grid\examples\workpool\. There are two Java programs:

MonteCarloPiModule.java
RunMonteCarloPiModule.java

MonteCarloPiModule.java. MonteCarloPiModule.java implements the interface for the workpool

```
package edu.uncc.grid.example.workpool;
import java.util.Random;
import java.util.logging.Level;
import edu.uncc.grid.pgaf.datamodules.Data;
import edu.uncc.grid.pgaf.datamodules.DataMap;
import edu.uncc.grid.pgaf.interfaces.basic.Workpool;
import edu.uncc.grid.pgaf.p2p.Node;

public class MonteCarloPiModule extends Workpool {
	private static final long serialVersionUID = 1L;
	private static final int DoubleDataSize = 1000;
	double total;
	int random_samples;
	Random R;
	public MonteCarloPiModule() {
		R = new Random();
	}
	@Override
	public void initializeModule(String[] args) {
		total = 0;
		Node.getLog().setLevel(Level.WARNING); // reduce verbosity for logging
		random_samples = 3000; // set number of random samples
	}
	public Data Compute (Data data) {
		DataMap<String, Object> input = (DataMap<String,Object>)data; //input gets data produced by DiffuseData()
		DataMap<String, Object> output = new DataMap<String, Object>(); // output will emit partial answers by method
		Long seed = (Long) input.get("seed"); // get random seed
		Random r = new Random();
		r.setSeed(seed);
		Long inside = 0L;
		for (int i = 0; i < DoubleDataSize ; i++) {
			double x = r.nextDouble();
			double y = r.nextDouble();
			double dist = x * x + y * y;
			if (dist <= 1.0) {
			  ++inside;
			}
		}
		output.put("inside", inside);	// store partial answer to return to GatherData()
		return output;
	}
	public Data DiffuseData (int segment) {
		DataMap<String, Object> d =new DataMap<String, Object>();
		d.put("seed", R.nextLong());
		return d; // returns a random seed for each job unit
	}
	public void GatherData (int segment, Data dat) {
		DataMap<String,Object> out = (DataMap<String,Object>) dat;
		Long inside = (Long) out.get("inside");
		total += inside; // aggregate answer from all the worker nodes.
	}
	public double getPi() { // returns value of pi based on the job done by all the workers
```

```
            double pi = (total / (random_samples * DoubleDataSize)) * 4;
            return pi;
      }
      public int getDataCount() {
            return random_samples;
      }
}
```

MonteCarloPiModule.java

In MonteCarloPiModule.java, two important classes are imported called Data and DataMap. Data is used to pass data between the master and slaves and is used with input parameters of the Compute and GatherData methods. DataMap is used within DiffuseData, Compute Data, and GatherData methods for specifying the data being passed and uses two parameters, a string and an object (generic typing). The first parameter can be any programmer chosen string and used to identify the second stored item.[3] DiffuseData method (executed by the master) creates a DataMap object and returns it with random seed for each job. The Compute method (executed by slaves) picks up the data from DiffuseData and creates a DataMap object for holding its partial results

RunMonteCarloPiModule.java. RunMonteCarloPiModule.java deploys the Seeds pattern and runs the workpool:

```
package edu.uncc.grid.example.workpool;

import java.io.IOException;

import net.jxta.pipe.PipeID;
import edu.uncc.grid.pgaf.Anchor;
import edu.uncc.grid.pgaf.Operand;
import edu.uncc.grid.pgaf.Seeds;
import edu.uncc.grid.pgaf.p2p.Types;

public class RunMonteCarloPiModule {

      public static void main(String[] args) {
            try {
                  MonteCarloPiModule pi = new MonteCarloPiModule();
                  Seeds.start( args[0] , false);

                  PipeID id = Seeds.startPattern(
                        new Operand( (String[])null, new Anchor( args[1]  ,  Types.DataFlowRoll.SINK_SOURCE), pi ) );
                  System.out.println(id.toString() );
                  Seeds.waitOnPattern(id);
                  System.out.println( "The result is: " + pi.getPi() ) ;

                  Seeds.stop();

            } catch (SecurityException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

RunMonteCarloPiModule.java

[3] DataMap extends Java HashMap.

Several classes are imported, PipeID, seeds-specific Anthor, Operand, Seeds, and Types. An instance of MonteCarloPiModule is first created. Seeds is started and deployed on the list servers using the Seeds method start, which takes as its first argument the path to the seeds folder on the local computer. In the code given, the path is provided as the string argument of main (first command line argument, args[0]). The Seeds method startPattern starts the workpool pattern on the computers. It requires as a single argument an Operand object. Creating an Object object requires three arguments. The first is a String list of argument that will be submitted to the remote hosts. The second is an Anchor object specifying the nodes that should have source and sink nodes (the master in this case) which in this provided as the string argument of main (second command line argument, args[1]). The third argument is an instance of MonteCarloPiModule previously created. The Seeds method waitOn Pattern waits for the pattern to complete, after which the results are obtained using the getPi method in MonteCarloPiModule.java. Seeds is stopped using the method stop. As mentioned above, to run this code, we will need to provide two command line arguments, the local path to the Seeds folder and the name of the local host. Both could have been hardcoded.

Executing the π program. The program can be executed on the command line or through an IDE. We choose here to use Eclipse.

Step 1. Open Eclipse

Start Eclipse and go to the workbench:

Step 2 Create a new project

Go to File -> New -> Project and select “Java Project”, or File -> New -> Java Project. Provide a name for the project, say PiApprox.

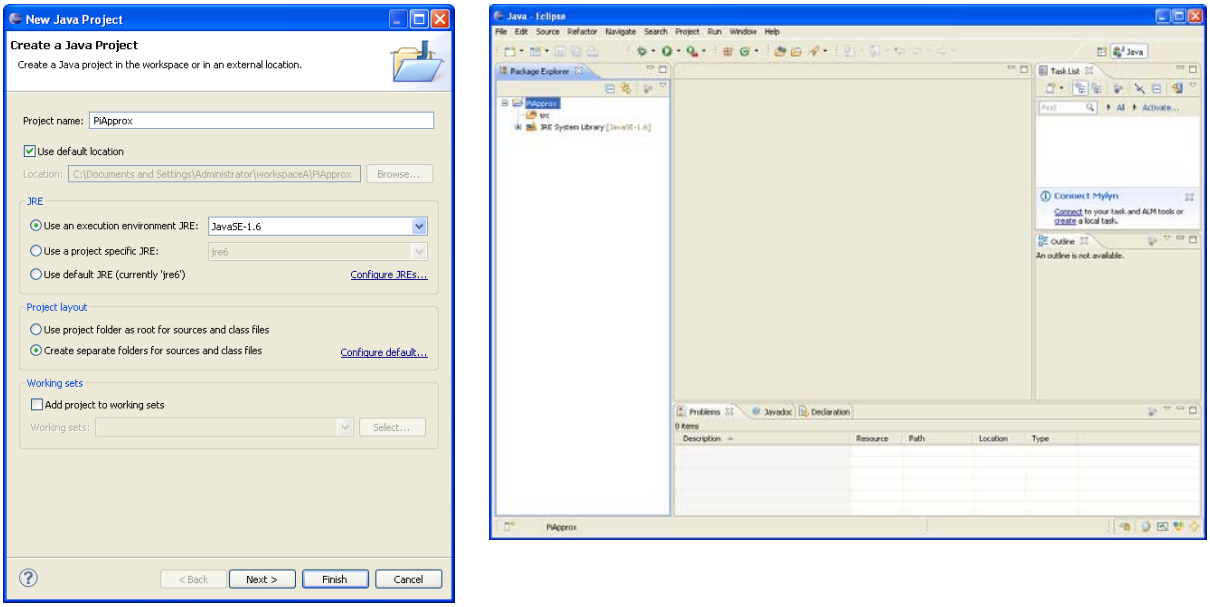


Figure 2 Creating a new Java project

If you click Finish at this time, you will see a new project created on the left panel as shown on the right image of Figure 2.

Step 5 Add source code - (May be an easier way to do this)

Go to New -> Class. For each source program you downloaded (MonteCarloPiModule.java and RunMonteCarloPiModule.java) fill the package edu.uncc.edu.example.workpool and source name (without the .java extension) and click Finish to add to your project (Figure 3). You should now see the source file structure for the project. Copy and paste the downloaded code into each source, replacing the template code. At this point, you will have unresolved errors as we have not yet added the paths to the libraries.

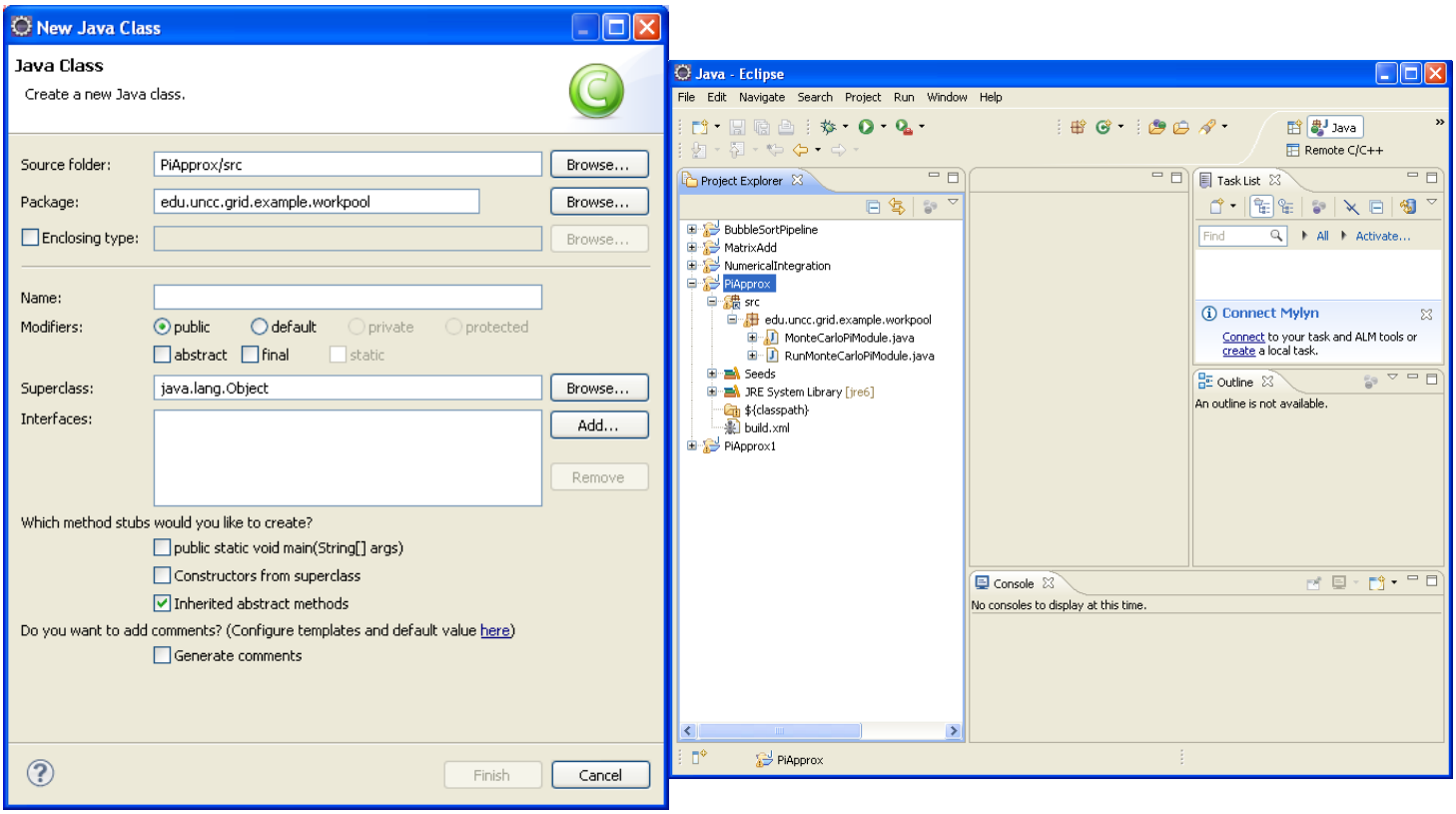


Figure 3 Adding source code

Step 4 Add path to Seeds libraries

Right click the PiApprox folder and select Properties. Select “Java Build Path” and the libraries tab. Select the Add library button and select User Library. Provide a name for the libraries, Seeds. Click on “Add Jars” (or “Add External Jars” depending upon the Eclipse version) and navigate through your file system to the Seeds lib folder. Select all the jars inside lib (control-A). Click “Open” (Figure 4) Finally you should see something like Figure 5. Click “OK”, and get back to the workbench (“Finish”, “OK”). At this point all unresolved references should vanish.

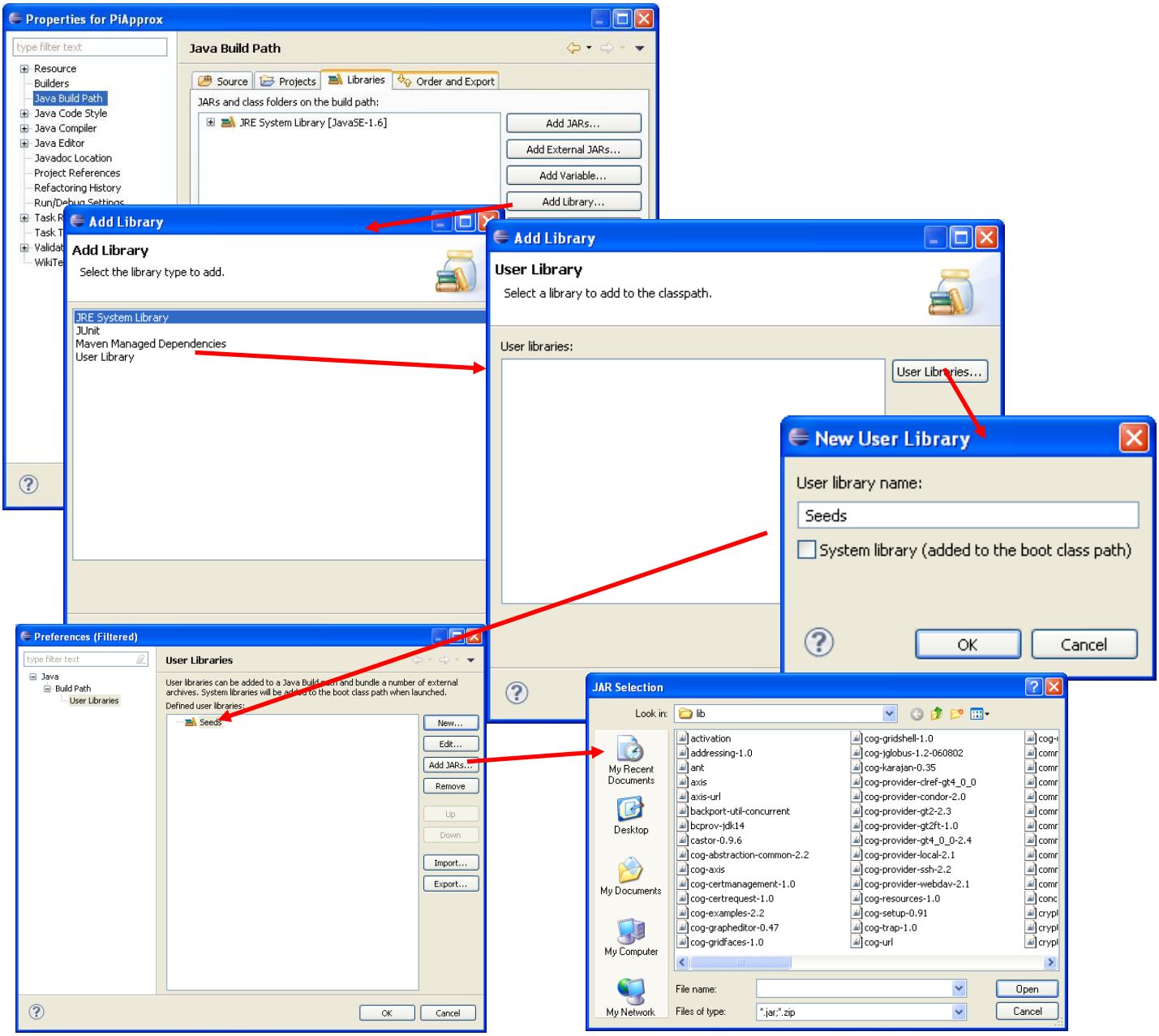


Figure 4 Setting Java Build path

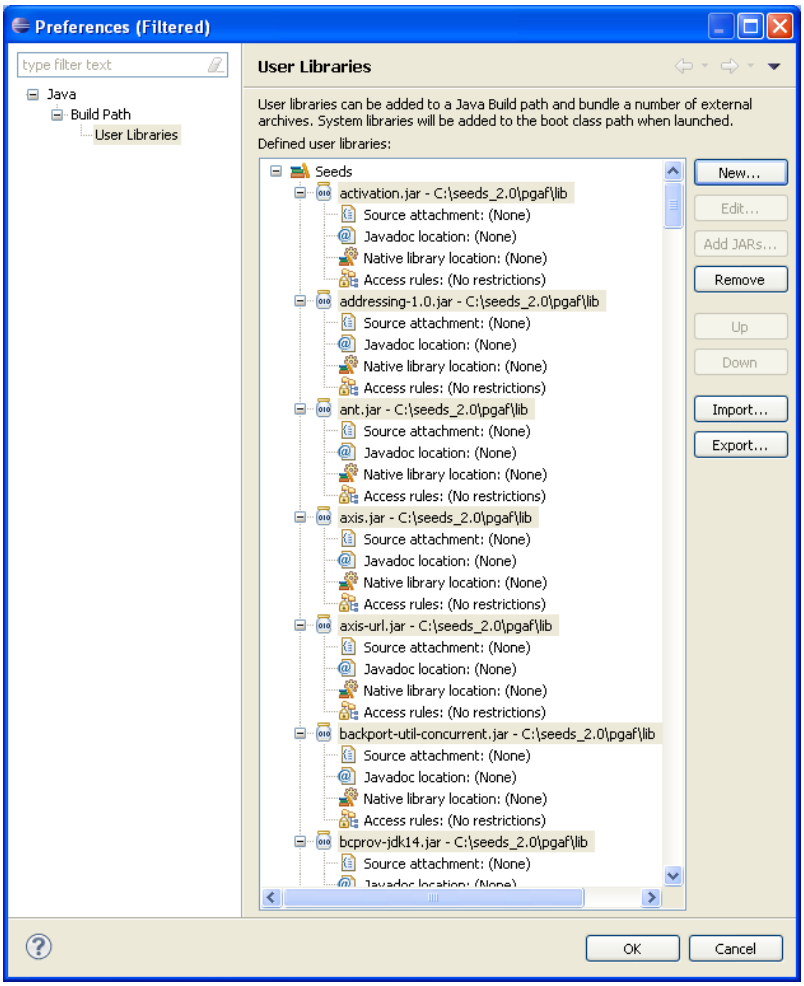
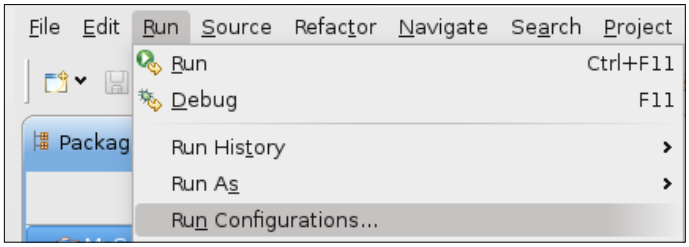


Figure 5 Java Build path set up

For subsequent Seeds projects, you will be able to simply select the named Seeds libraries.

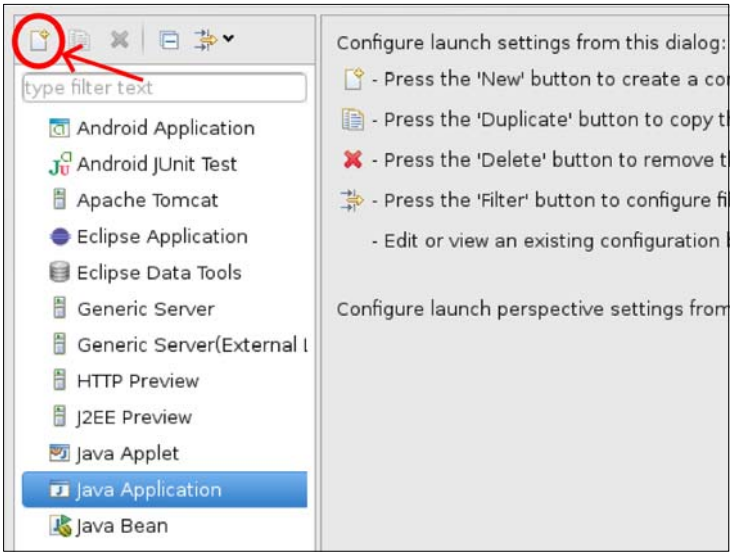
Step 5 Run program

Go to **Run -> Run Configurations** …



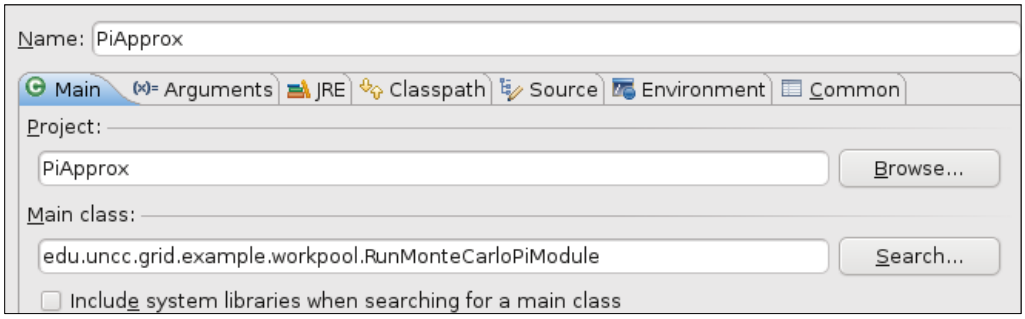
Click the new launch configuration button.

Name the Run Configuration PiApprox.



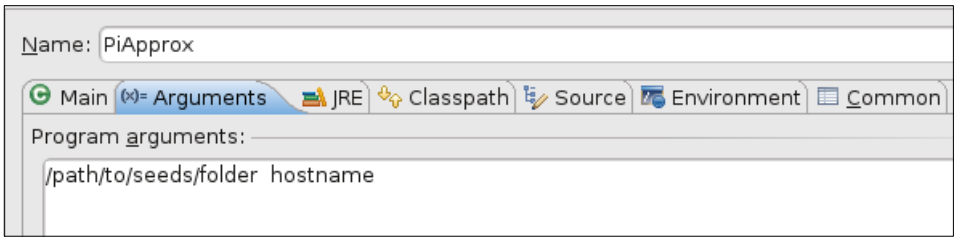
On the Project field, make sure the PiApprox project is selected. If not, click the Browse button. The dialog window should have the PiApprox project among the choices.

Main class. Fill in the Main Class field. Click the Search... button, and start typing RunMonteCarloPiModule. The choice list should narrow down as you continue typing.



Select the class RunMonteCarloPiModule and click Apply.

Arguments. Now, click the tab named (x)=Arguments. The first argument should be the absolute path to the seeds folder, for example "C:\Program Files\seed_2.0\pgaf". Includes double quotes to make a string if there are one or more spaces in the path. The second argument is the name of the host computer . To get this, you can type hostname command from the command line. The name should be the same as you put in AvailableServers.txt in Task 2.



Go back the Eclipse Run Configurations dialog and click Run. The red text is log code for the JXTA platform, which is used by the Seeds framework. The red and black output should include the end result for π in black (and then continue with more output before terminating).

How many random numbers were tried by the π approximation program?

Task 4 (20%)

Write and test a workpool program on the Framework to multiply two 3 x 3 matrices. Add code to measure the execution time.

Grading

Every task and subtask specified will be allocated a score so make sure you clearly identify each part you did.

Assignment Submission

Produce a document that show that you successfully followed the instructions and performs all tasks by taking screen shots and include these screen shots in the document. Give at least one screen shot for each numbered task in each exercise. (To include screen shots from Windows XP, select window, press Alt-Printscreen, and paste to file.) Provide insightful conclusions. Submit one pdf file containing everything by the due date as described on the course home page. Include all code, not as screen shots of the listings but complete properly documented code listing.