Google Web Toolkit – Creating/using external JAR files

If you develop some code that can be reused in more than one project, one way to create a module is to create an external JAR file. This JAR file can be included in the build path for multiple projects.

Required folders/files

The JAR file will need to have the following folders/files:

Assume that you have a Project named `MyProj`, with a package name of `mydomain.gwt`. Also assume that you have the following source files (in your client directory), Test.java, and Test2.java. Your JAR file should contain the following:

```
META-INF/
META-INF/MANIFEST.MF
mydomain/
mydomain/gwt
mydomain/gwt/client
mydomain/gwt/client/Test.java
mydomain/gwt/client/Test.class
mydomain/gwt/client/Test2.java
mydomain/gwt/client/Test2.class
mydomain/gwt/MyProj.gwt.xml
```

Both the source (.java) and the bytecode (.class) files should be in the `client` folder. (They could be stored in other folders if the appropriate changes are made to the module.gwt.xml file, but this is not usually done.) The source files must be present to be compiled to JavaScript.

Format for source files

Assuming the project and package structure above, the source file should have the following format:

```
package mydomain.gwt.client;

// import statements

public class Test
{
   // code goes here
}
```

Contents of MyProj.gwt.xml

The module.gwt.xml file should not contain any entry points. All it needs is the inherits for any modules that are used. For example:

```
<module>
```

```
  <inherits name='com.google.gwt.user.User'/>
</module>
```

Using the external JAR

To use the external JAR file, you modify the project's Java Build Path, by going to the Libraries tab, selecting External JARs, and selecting the JAR file that you create.

In addition, in your module.gwt.xml file, you need to reference the project file as in:

```
<module>
 <!-- Inherit the core Web Toolkit stuff -->
 <inherits name="'com.google.gwt.user.User'/">

 <!-- ** ADD THIS ** -->
 <inherits name="'mydomain.gwt.MyProj'/">

 <!-- Specify the app entry point class. -->
 <entry-point
     class="'mydomain.gwt.client.MyApplication/">
</module>
```

Example: GWTCanvasHelpers

To give a clearer idea of how to do this, let's go through an actual example. In this case, let's create a module that will help to draw arrows on a GWTCanvas object.

In Eclipse, go to File=>New=>Web Application Project. Choose “GWTCanvasHelpers” for the project name, and “edu.hawaii.takebaya.gwt” for the package name. Deselect the Google App Engine checkbox.

In the src folder, delete all the Greeting Service code that the GWT plugin inserts. That is, go to the src/edu/hawaii/takebaya/gwt/client folder and delete GreetingService.java and GreetingServiceAsync.java. Also, go to src/edu/hawaii/takebaya/gwt/server and delete GreetingServiceImpl.java.

Modify GWTCanvasHelper.gwt.xml to contain:

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='gwtcanvashelpers'>
  <inherits name='com.google.gwt.widgetideas.GWTCanvas'/>
</module>
```

In the src/edu/hawaii/takebaya/gwt/client folder, change the contents of GWTCanvasHelpers.java to the following:

```
package edu.hawaii.takebaya.gwt.client;

import com.google.gwt.widgetideas.graphics.client.GWTCanvas;

public class GWTCanvasHelpers
{
   public static void drawLine(Point a, Point b, GWTCanvas canvas)
   {
      canvas.beginPath();
      canvas.moveTo(a.x, a.y);
      canvas.lineTo(b.x, b.y);
      canvas.stroke();
   }
   public static void drawArrow(Point a, Point b, GWTCanvas canvas)
   {
      double deltaX = b.x - a.x;
      double deltaY = a.y - b.y;
      double dirRadians = Math.atan2(deltaY, deltaX);
      double dirDegrees = dirRadians*180.0/Math.PI;
      double dirOpposite = dirDegrees + 180.0;
      while (dirOpposite > 360) {
         dirOpposite = dirOpposite - 360;
      }
      while (dirOpposite < 0) {
         dirOpposite = dirOpposite + 360;
      }
      double deltaX1 = 10*Math.cos((dirOpposite+20)*Math.PI/180.0);
      double deltaY1 = 10*Math.sin((dirOpposite+20)*Math.PI/180.0);
      double deltaX2 = 10*Math.cos((dirOpposite-20)*Math.PI/180.0);
      double deltaY2 = 10*Math.sin((dirOpposite-20)*Math.PI/180.0);
      Point newPoint1 = new Point(b.x+deltaX1,b.y-deltaY1);
      Point newPoint2 = new Point(b.x+deltaX2,b.y-deltaY2);
      drawLine(a,b,canvas);
      drawLine(b,newPoint1,canvas);
      drawLine(b,newPoint2,canvas);
   }
}
```

This class will require the Point class. Right-click on the src/edu/hawaii/takebaya/gwt/client folder and select New=>Class. Name the class Point, and make it have the following contents:

```
package edu.hawaii.takebaya.gwt.client;

public class Point
{
    public double x;
    public double y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Creating the JAR file using Eclipse

Right-click on the project in the Package Explorer. Select Export=>Java=>JAR file. See Figure 1 on page 4.

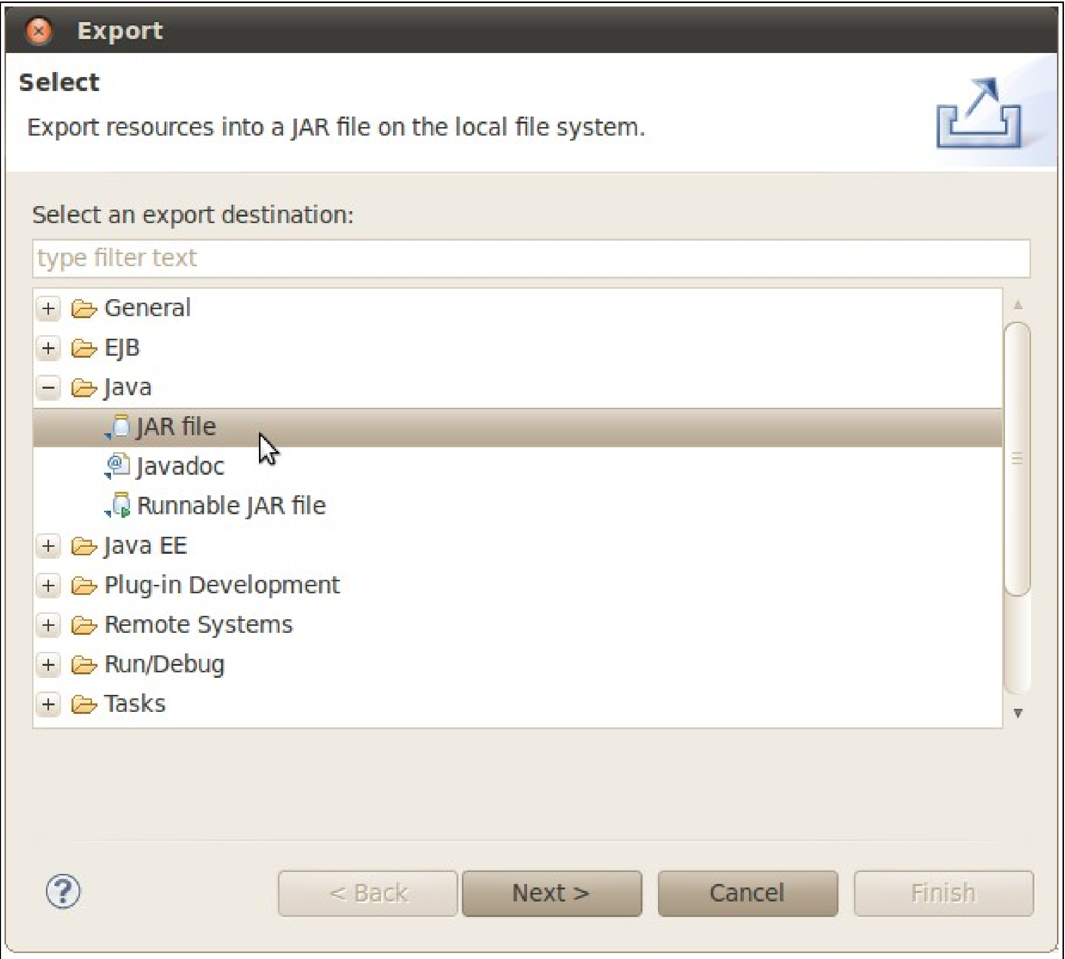
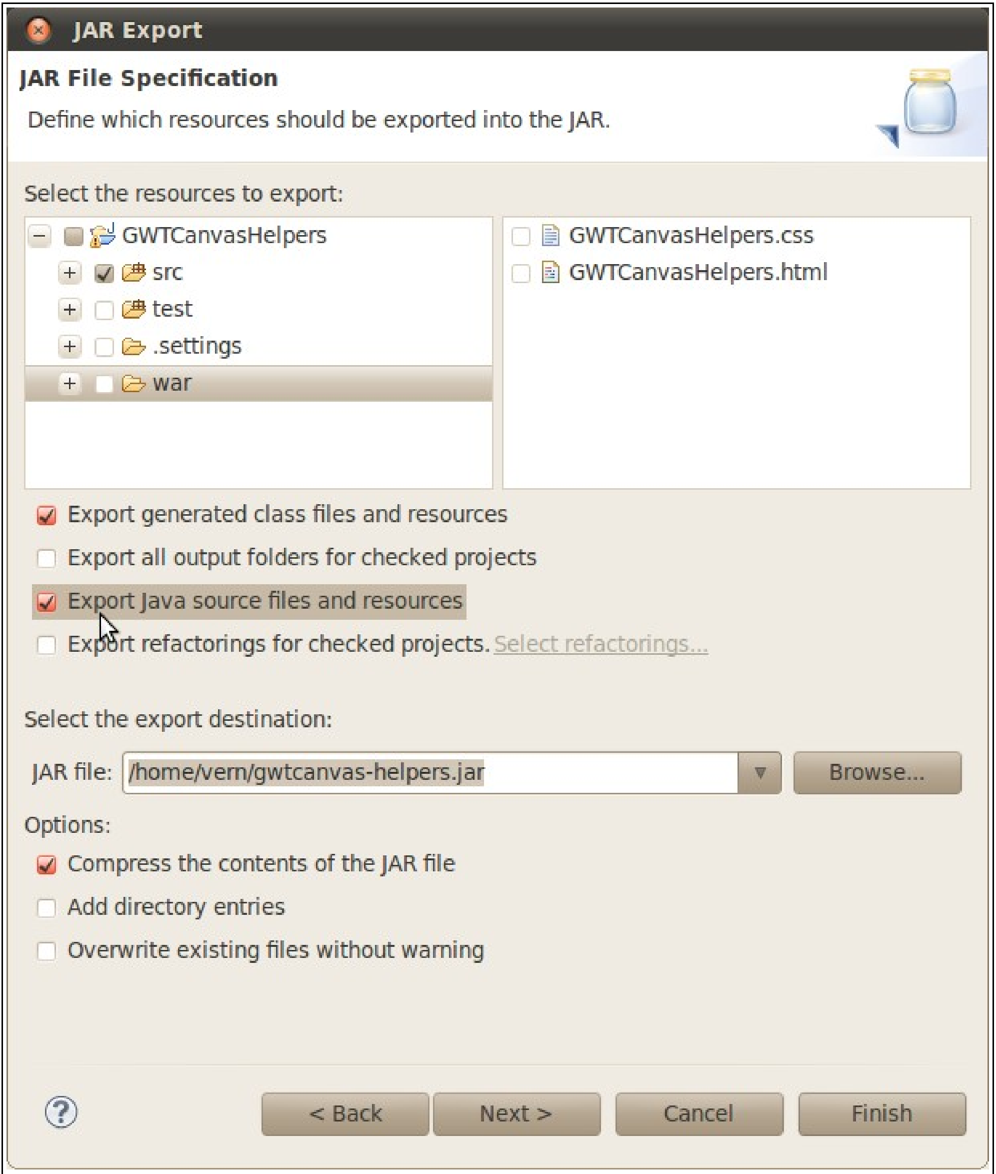


Figure 1: Screen shot showing start of procedure to export to JAR.

Click Next. As show in Figure 2 on page 5, Turn off all the check boxes except for src for the resources to export. Turn on the checkbox “Export Java source files and resources” so that the Java source code is included in the JAR file. Select the export destination and filename. Then, click Next.



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [–] GWTCanvasHelpers
  - [✓] src
  - [ ] test
  - [ ] .settings
  - [ ] war

- [ ] GWTCanvasHelpers.css
- [ ] GWTCanvasHelpers.html

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [✓] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

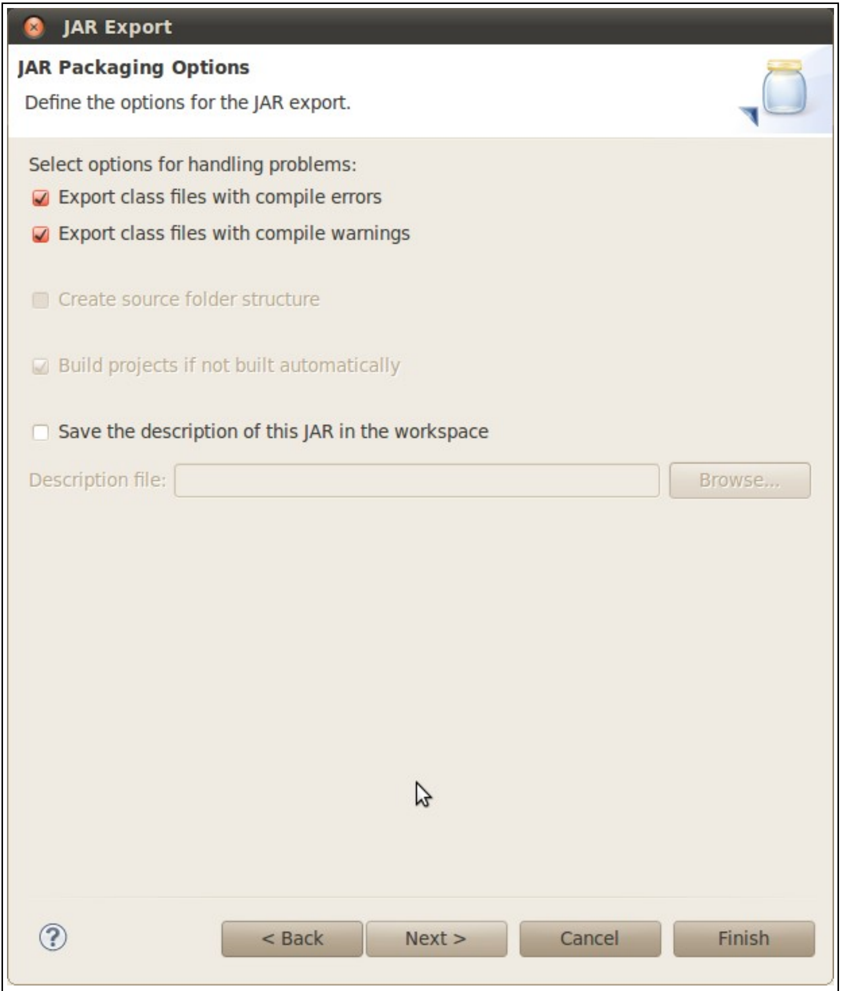
JAR file: /home/vern/gwtcanvas-helpers.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

Figure 2: Screen shot showing resources to export to JAR.

After clicking Next, the following page is shown. See Figure 3 on page 6. Keep the default options of exporting clas files with compile errors and warnings as shown.



JAR Export

JAR Packaging Options

Define the options for the JAR export.

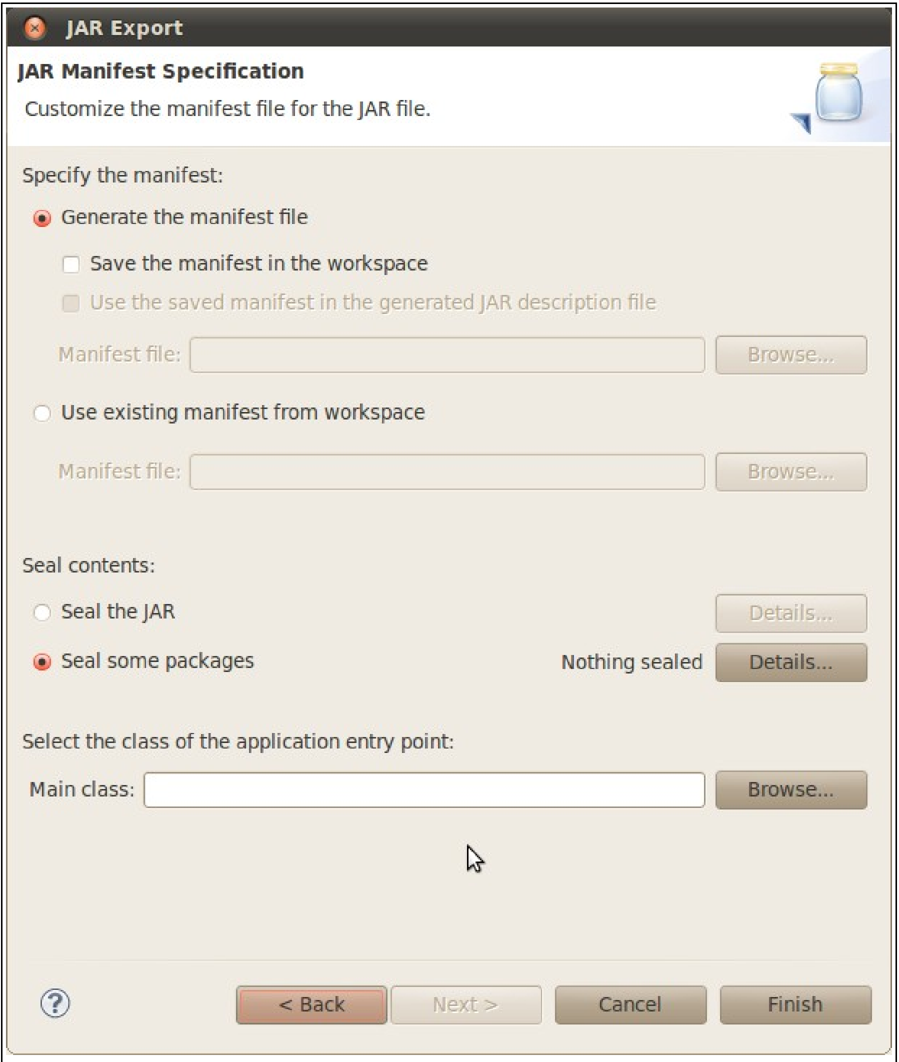
Select options for handling problems:

- [✓] Export class files with compile errors
- [✓] Export class files with compile warnings
- [ ] Create source folder structure
- [✓] Build projects if not built automatically
- [ ] Save the description of this JAR in the workspace

Description file:

Figure 3: Screen shot showing the JAR packaging options.

Click Next. As shown in Figure 4 on page 7, select "Generate the manifest file". Since this is not an executable jar, do not set the Main class. Click Finish.


JAR Export

JAR Manifest Specification

Customize the manifest file for the JAR file.

Specify the manifest:

- [✓] Generate the manifest file
  - [ ] Save the manifest in the workspace
  - [ ] Use the saved manifest in the generated JAR description file
  - Manifest file:
- [ ] Use existing manifest from workspace
  - Manifest file:

Seal contents:

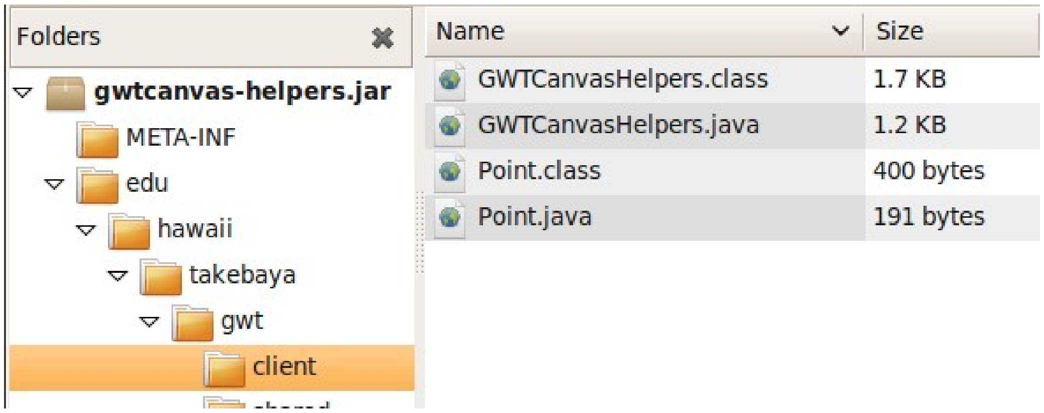
- [ ] Seal the JAR
- [✓] Seal some packages — Nothing sealed

Select the class of the application entry point:

Main class:

Figure 4: Screen shot showing JAR manifest specification.

The resulting JAR file looks like this:


Folders: gwtcanvas-helpers.jar / META-INF / edu / hawaii / takebaya / gwt / client

Name	Size
GWTCanvasHelpers.class	1.7 KB
GWTCanvasHelpers.java	1.2 KB
Point.class	400 bytes
Point.java	191 bytes

Using the JAR file in another project

The following is a simple example of using this JAR file in a GWT project. In Eclipse, start by going to File=>New=>Web Application Project. Set the project name to “TestDraw”, and the package name to “vern.test”. Turn off the Google App Engine checkbox.

Delete all the references to the Greeting Servlet. That is delete all the Java files that are put in by the plugin that contain Servlet (GreetingServlet.java, GreetingServletAsync.java, and GreetingServletImpl.java).

Modify “TestDraw.gwt.xml” to contain:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <module rename-to='testdraw'>
3    <!-- Inherit the core Web Toolkit stuff. -->
4    <inherits name='com.google.gwt.user.User'/>
5    <inherits name='com.google.gwt.widgetideas.GWTCanvas'/>
6    <inherits name='edu.hawaii.takebaya.gwt.GWTCanvasHelpers'/>
7
8    <!-- Inherit the default GWT style sheet.  -->
9    <inherits name='com.google.gwt.user.theme.standard.Standard'/>
10   <!-- Specify the app entry point class.     -->
11   <entry-point class='vern.text.client.TestDraw'/>
12 </module>
```

The key line (shown in bold) is line 6. Note how the package name followed by the module name is how we reference the module.

Modify “TestDraw.html” to contain:

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <link type="text/css" rel="stylesheet" href="TestDraw.css">
    <title>Demo of GWTCanvasHelpers</title>
    <script type="text/javascript" language="javascript"
      src="testdraw/testdraw.nocache.js"></script>
  </head>
  <body>
    <!-- RECOMMENDED if JavaScript must be enabled -->
    <noscript>
      <div style="width: 22em; position: absolute; left: 50%;
        margin-left: -11em; color: red; background-color: white;
        border: 1px solid red; padding: 4px; font-family: sans-serif">
        Your web browser must have JavaScript enabled
        in order for this application to display correctly.
      </div>
    </noscript>
    <h1>Test of GWTCanvasHelpers</h1>
  </body>
</html>
```

Modify “TestDraw.css” to contain:

```
/** Add css rules here for your application. */


/** Example rules used by the template application (remove for your
app) */
h1 {
  font-size: 2em;
}
```

Modify “web.xml” to contain:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>TestDraw.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Add the JAR file, “gwtcanvas-helpers.jar” to the project's Java Build Path property. Add the gwt-incubator-2010204-r1747.jar file as well.

Finally, modify “TestDraw.java” to contain the following:

```
1  package vern.text.client;
2  
3  import com.google.gwt.core.client.EntryPoint;
4  import com.google.gwt.user.client.ui.RootPanel;
5  import com.google.gwt.user.client.ui.VerticalPanel;
6  import com.google.gwt.widgetideas.graphics.client.GWTCanvas;
7  import edu.hawaii.takebaya.gwt.client.GWTCanvasHelpers;
8  import edu.hawaii.takebaya.gwt.client.Point;
9  
10 public class TestDraw implements EntryPoint
11 {
12    public void onModuleLoad()
13    {
14       VerticalPanel mainPanel = new VerticalPanel();
15       GWTCanvas canvas = new GWTCanvas(400,400);
16       mainPanel.add(canvas);
17       Point from = new Point(30,30);
18       Point to = new Point(100,100);
19       GWTCanvasHelpers.drawArrow(from, to, canvas);
20       Point next = new Point(200,50);
21       GWTCanvasHelpers.drawArrow(to, next, canvas);
22       RootPanel.get().add(mainPanel);
23    }
24 }
```

Note on lines 7 and 8 how we import the classes for use. Lines 19 and 21 draw the arrows using the Points that are set up on lines 17, 18, and 20. The resulting web application has the following appearance:

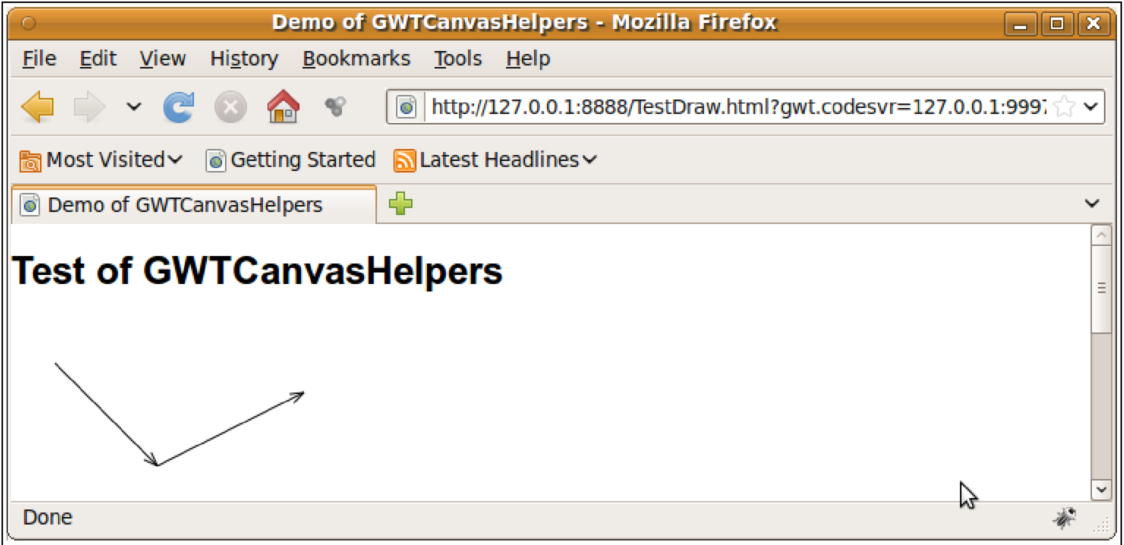

Demo of GWTCanvasHelpers - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:8888/TestDraw.html?gwt.codesvr=127.0.0.1:9997
Most Visited
Getting Started
Latest Headlines
Demo of GWTCanvasHelpers
Test of GWTCanvasHelpers
Done

Figure 5: Screen shot showing simple web application that makes use of drawArrow() method.