Introduction to Java (part 2)

Jules White

Bradley Dept. of Electrical and
Computer Engineering
Virginia Tech
Jules.white@vt.edu

www.dre.vanderbilt.edu/~jules

Exception Handling

- If an error condition occurs, a method can throw an “Exception” to notify the caller of the error
- An Exception causes the normal program execution to stop and for the Exception handler currently in scope to get called

```
//the class we are going to listen to
public class MyApplication{

  public void catchAnException(){
     //try is used to mark a block of code that
     //you would like to catch exceptions from
     try {
         //do something bad to throw an exception
         MyApplication app = null;
         app.catchAnException();
     }
     //every try block has a catch block that handles the exceptions
     catch (Exception e){
        //do something with the exception
        e.printStackTrace();
     }
  }
}
```

Exception Handling

- A method can declare an exception that any caller MUST provide an exception handler for

```
//the class we are going to listen to
public class MyApplication{

  //to call this method, you will have to surround the
  //call with a try/catch block
  public String catchAnException() throws Exception{
      if(someBadSituationOccurs){
         throw new Exception(“Yikes!”);
      }

      return “maybe you got lucky”;
  }
}
```

Exception Handling

- Exception handlers can be specified for different types of exceptions
- If you don’t specify an Exception handler for the type of exception that gets thrown, the next exception handler up the chain will get called

```
//the class we are going to listen to
public class MyApplication{

  public void catchAnException(){
     try {
         …
     }
     catch (IOException io){
        …
     }
     catch (FooException e){
        //do something with the exception
        e.printStackTrace();
     }
  }
}
```

Exception Handling

- If you want to catch every possible Exception type, use “Exception
- To catch every possible type of error, use “Throwable”:

```
…
try{
   File f = new File(“foo”);
}
catch(Throwable e){
  //every possible recoverable runtime error will be caught
}
```

Exception Handling

- You can throw a RuntimeException without declaring it in a “throws” clause
- A RuntimeException can be constructed to wrap another exception:

```
…
try{
   File f = new File(“foo”);
}
catch(Exception e){
  throw new RuntimeException(e);
}
```

Exercise

```
//What is wrong with this code?

public class MyApplication{

  public String getFullName() {
     return getName()+” Foo”;
  }

  public String getName() throws Exception {
     …
  }
}
```

Exercise

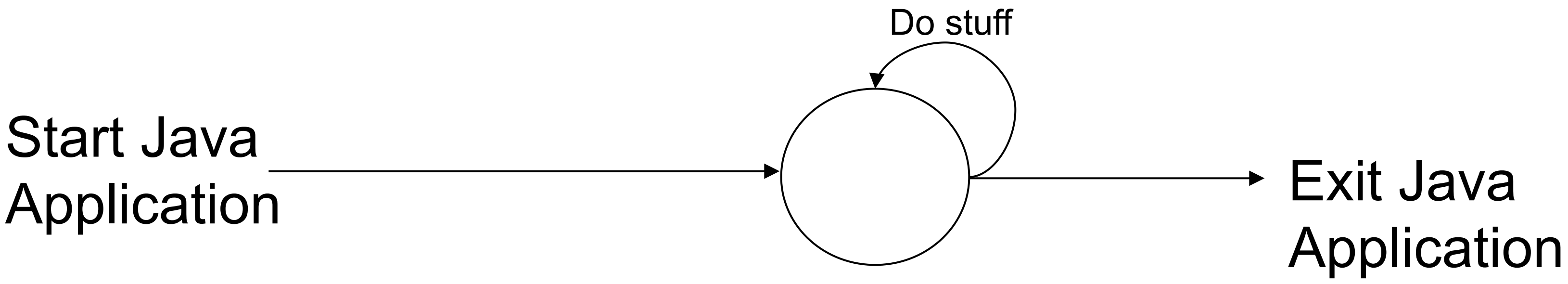
```
//Rewrite the getFullName() method so that you can eliminate
//the “throws Exception” declaration
//If an exception is thrown, your version of the method should pre
//prefix the name with “Exception” and print a stack trace

public class MyApplication{

  public String getFullName() throws Exception{
     return getName()+” Foo”;
  }

   public String getName() throws Exception {
     …
  }
}
```

Exercise

```
//Rewrite the getFullName() method so that you can eliminate
//the “throws Exception” declaration
//If an exception is thrown, your version of the method should
//rethrow the exception as a RuntimeException

public class MyApplication{

  public String getFullName() throws Exception{
     return getName()+” Foo”;
  }

  public String getName() throws Exception {
     …
  }
}
```

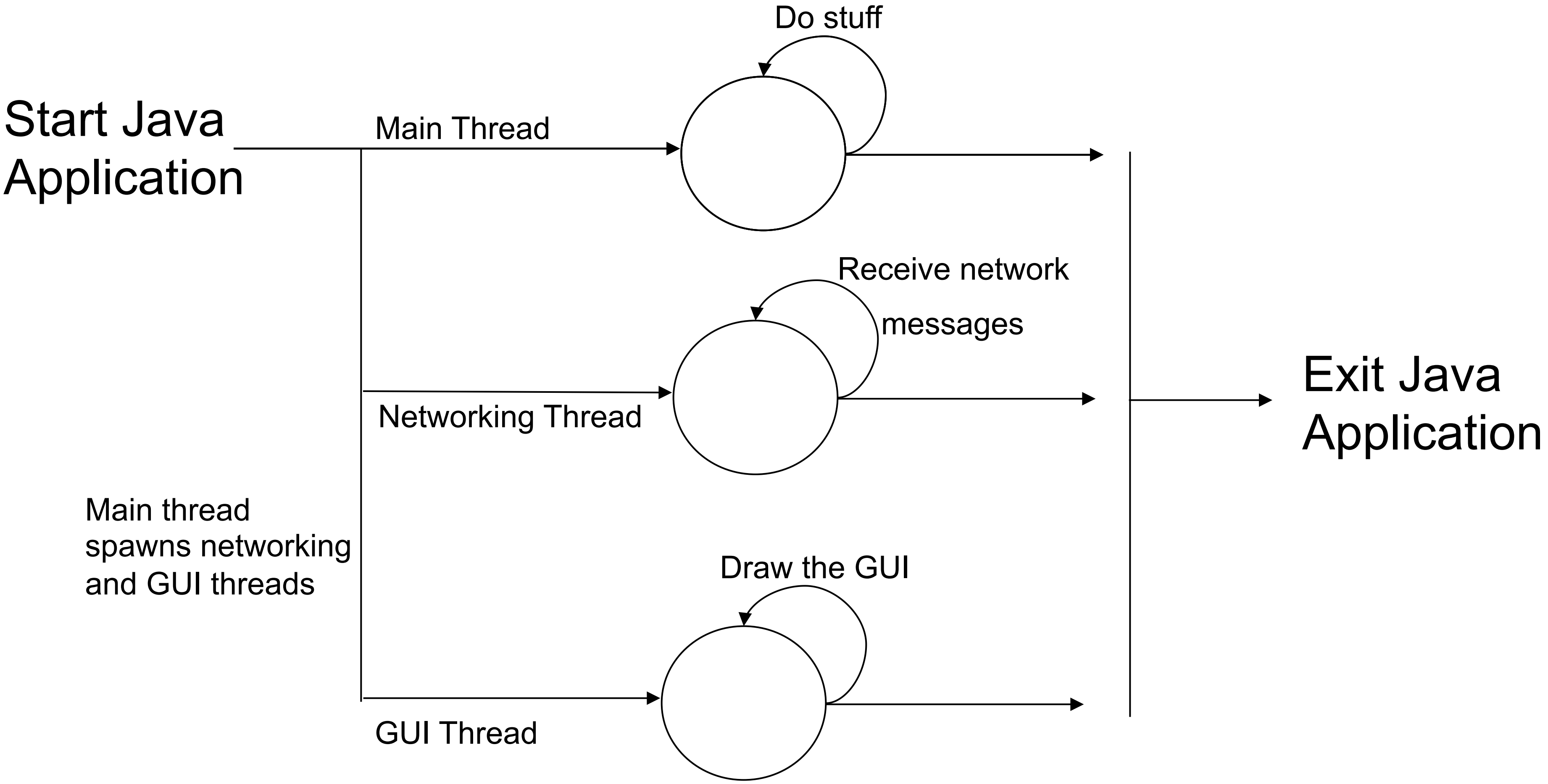
Threads

- Every application can have one or more threads of execution
- Each thread is a line of execution through the application
- Multiple threads in an application run in parallel
- Every Java application has one thread that serves as the main thread of execution
- A Java application will not exit as long as there is one active thread



Threads

- Most applications that include network communication or graphics involve multiple threads
- Example: The application can't stop updating the GUI just because it is waiting for a network message



Main Java Thread of Control

- In a standard application, your code owns the main thread of control
- Your application defines a static method (meaning it can be accessed without creating an instance of the containing class) called “main”
- The parameters to the method are the arguments to the program from the command line

```
//the class we are going to listen to
public class MyApplication{

  public static void main(String[] args){
      while(somecondition){
               //do stuff
             }

      //exit the application
    }

}
```

Framework

- A framework is a set of code that controls the main thread of execution for you
- You provide code that plugs into the framework
- A framework uses “inversion of control”, meaning “don’t call me, I will call you”
- The framework orchestrates and coordinates the parts of the application

```
public class MyApplication{
    //the framework decides when to call your code to do stuff
    public void doStuff(){…}

  public static void main(String[] args){
      Framework someFramework = new Framework(new MyApplication());

            //no while loop, the framework orchestrates the application
            someFramework.start();

            //exit the application when the framework decides to stop
  the application
    }
}
```

Framework

- A framework is incomplete, you must provide some code
- Frameworks are designed to capture common code that is used across a broad class of applications and make it easily reusable
- Frameworks provide structure for your application, they typically dictate the architecture of the application and the software patterns used
- Frameworks always provide mechanisms for extension
  - e.g. Abstract classes / interfaces for your to implement

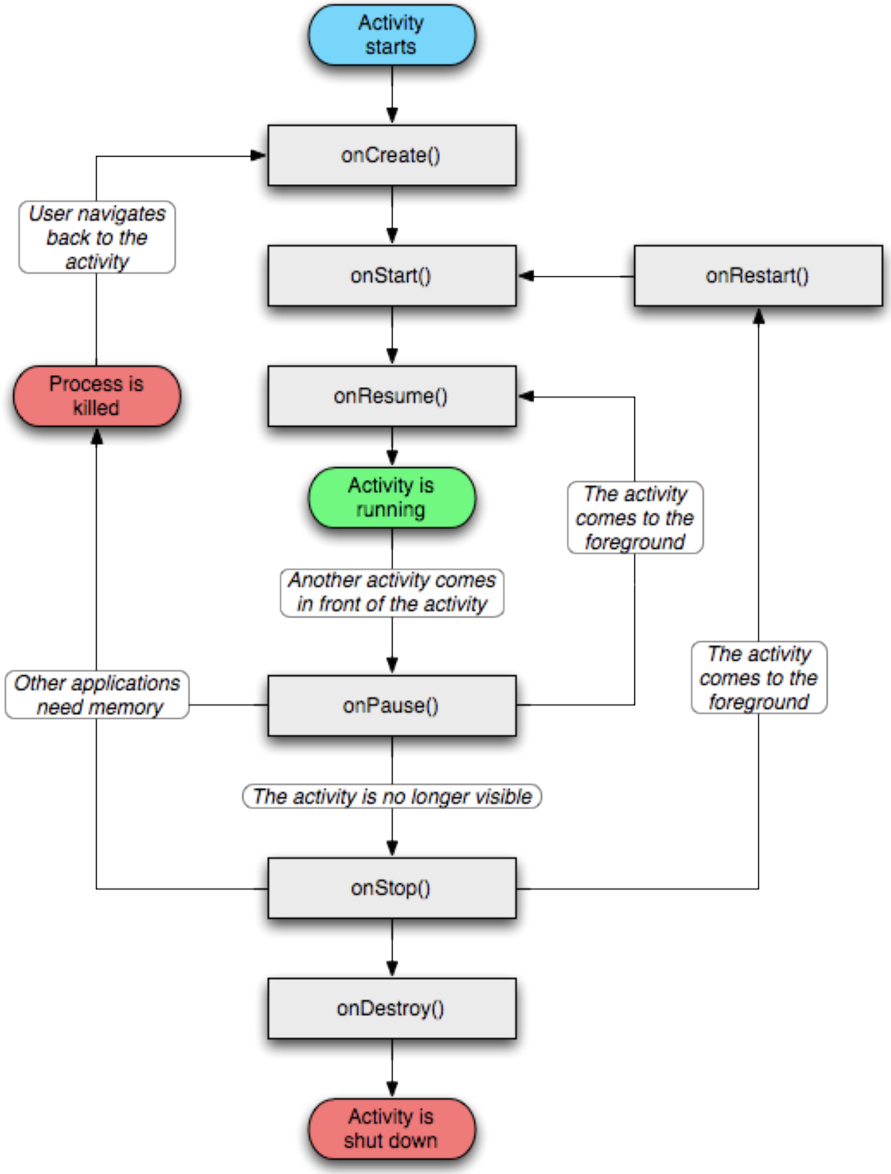
- Frameworks are common in networked, web, and GUI applications
  - Java Spring Framework
  - Enterprise Java Beans Framework
  - Google App Engine Framework
  - Android Framework
  - .Net Framework

Framework

- Most (if not all) of the code you write in this class will use a framework
- Some of your code may implement your own framework
- All Android or iPhone apps run inside of a framework

Activity Life-cycle

- The Android framework controls the main thread of control
- Your code is invoked in the onCreate(), onStart(), onResume(), and other life-cycle methods
- The framework decides when your application should stop, start, pause, etc.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Abstract Class / Interface Extension

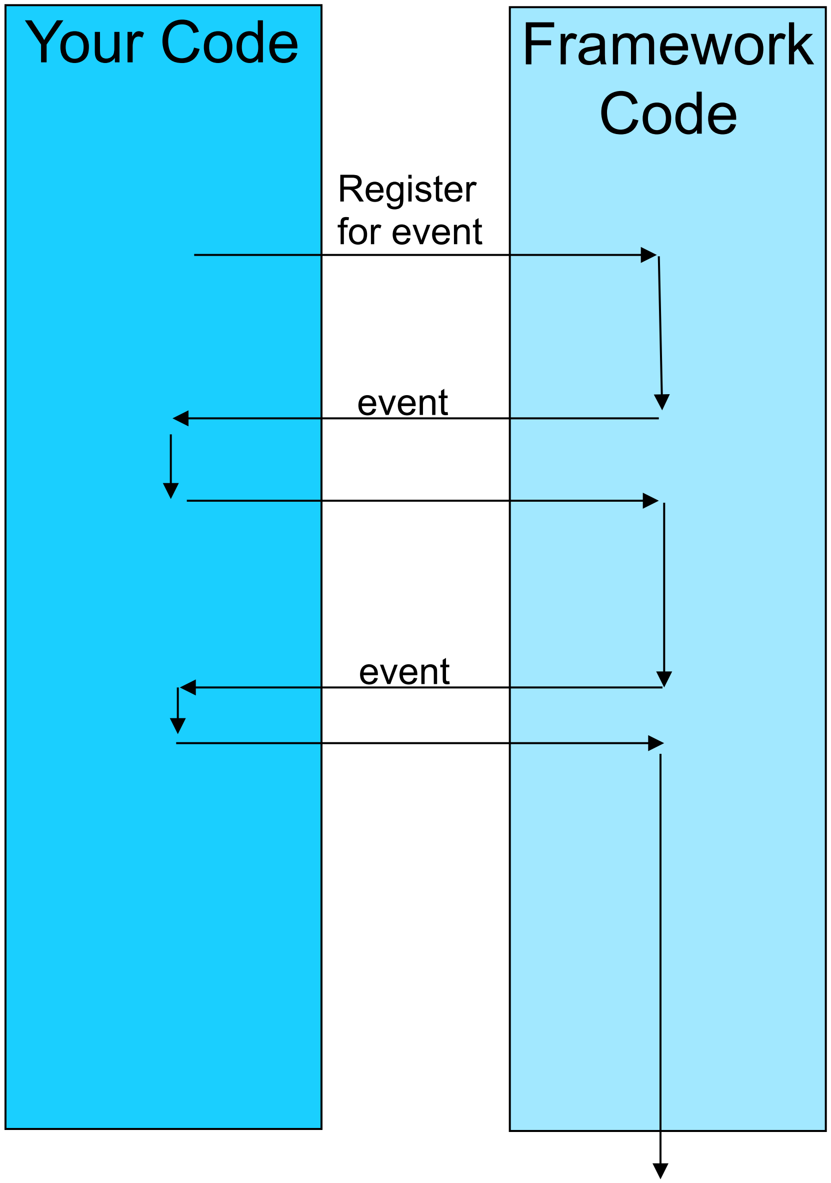
- Some frameworks use abstract classes or interfaces that you must extend to integrate your code
- You construct an instance of the framework and pass in your concrete implementations of the abstract classes
- Your code implements incomplete methods
- The main thread of control calls out to your code to fill in the holes

```
public abstract class MessageProcessor {
   public abstract void process(Message m);
}
```

```
public class NetworkFramework {
   private MessageProcessor processor_;
   public NetworkFramework(MessageProcessor m){ processor_ = m;}
   public void start() {
       while(running){
            Message m = receiveNetworkMessage();
            processor_.process(m);
       }
}
```

Event-driven Frameworks

- Many frameworks use an event-driven programming model to allow your code to plug into them
- Your code registers itself as a listener for specific types of events that can occur within the framework
  - Arrival of network messages
  - Clicks on GUI elements
- When an event you are registered for occurs, the framework temporarily turns the thread of execution over to your code
- When your code exits, the main thread of control returns to the framework



Event Listeners in Java

- Problem: I want to be notified whenever an event occurs but I don’t want the execution of the program to wait on that event. Example, I want to be notified when a button is clicked but I want to do other stuff until the click happens.
- Solution: Event Listeners
- How it works:
  - A listener interface is created that contains methods for sending events to an interested party
  - The class that supports listeners provides methods to add and remove listener objects that implement the interface
  - Classes that want to listen to the other class implement the listener interface and add themselves as listeners to instances of the subject class

Event Listeners in Java

- Example:

```
//The listener interface
public interface ClickListener {
   public void onClick(ClickEvent evt);
}
```

```
//the class we are going to listen to
public class Button{
   private ClickListener listener_;
   public void setClickListener(ClickListener l){listener_ = l;}

   //this method is controlled by the framework
   public void theButtonWasClicked(ClickEvent ce){
      //do some stuff inside the button

      //now tell the listener about the click
      listener_.onClick(ce);
   }
}
```

Event Listeners in Java

- Example:

```
//the class we are going to listen to
public class MyGui implements ClickListener{
   public void onClick(ClickEvent evt){
     //respond to the click event
   }

   public void createSubmitButton(){
       Button b = new Button(“submit”);

       //register to receive events from the button
       b.setClickListener(this);
   }
}
```

Exercise

- Write a an interface for a LocationManager. The LocationManager should allow clients to add listeners that can receive callbacks notifying them as the phone’s position changes.

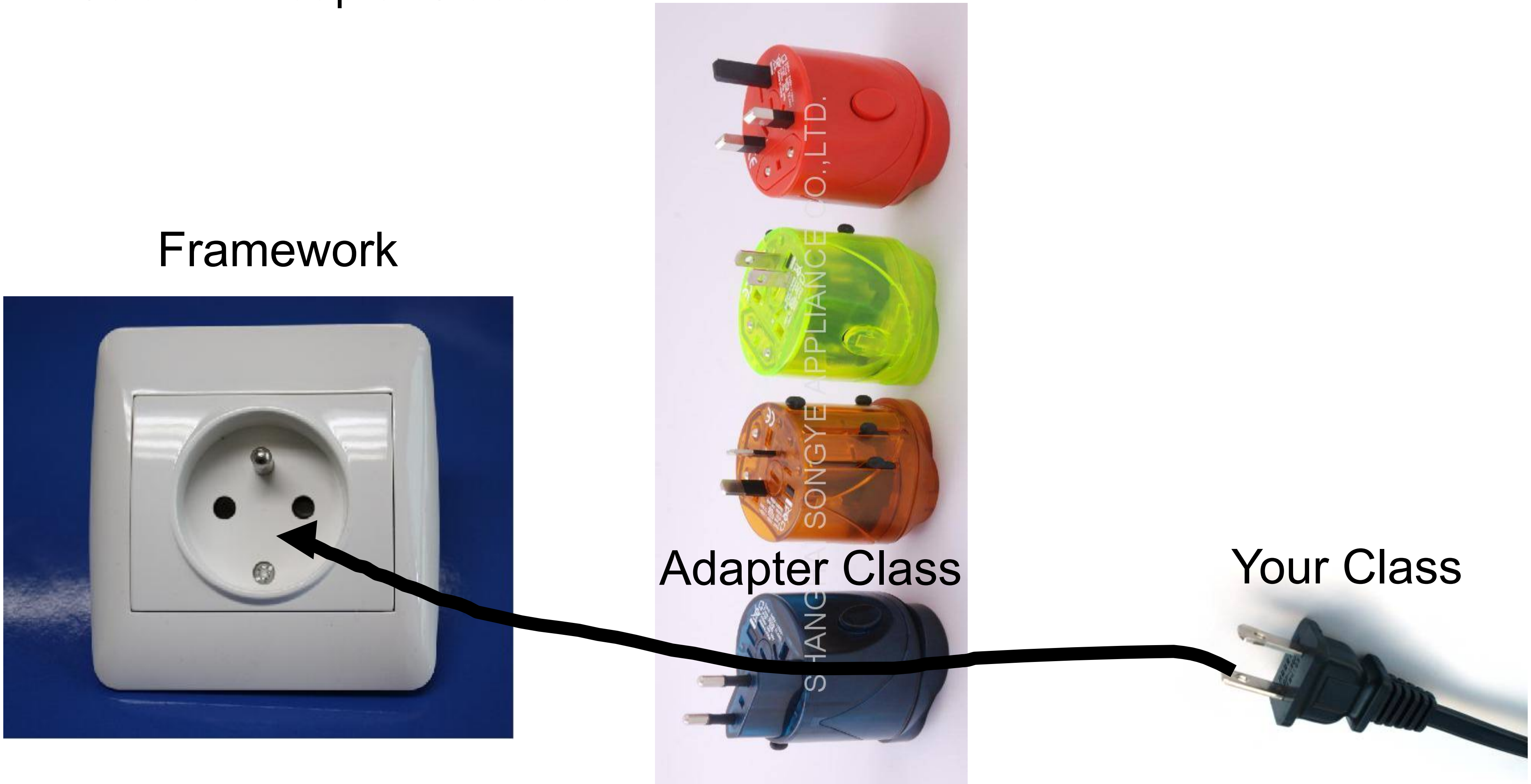
- You must provide a way for clients to specify the granularity of updates that they want (e.g. 1m, 10m, 100m, etc.)

- The listener interface that you create should have methods for being notified when the GPS signal is lost

- The client should have a mechanism for determining the current accuracy of any location fix returned to it

Adapters

- Problem: You need a way to plug your code into a framework
- Your top-level classes don’t inherit from abstract framework classes or implement any framework classes
- Solution: Adapter Classes



Framework

Adapter Class

Your Class

Java Inner Classes

- Problem: Sometimes you need a specialized class or data structure to simplify the implementation of one specific class. You don’t want to create a new top-level class. Moreover, you need the helper class to have direct access to the other class’ private variables.
- Java Solution: Inner Classes

- An inner class is a class defined within another class
- An inner class can access all of its parent class’ private variables and methods
- An inner class instance can typically only exist inside an instance of the enclosing class
- Inner classes make great adapters

Java Inner Class Adapters

- Example:

```
public class MyGui{
    //An inner class used as an adapter class
    private class ButtonListener implements ClickListener{

      public void onClick(ClickEvent evt){
               submit();
          }
      }
   }

   private void createSubmitButton(){
      Button b = new Button(“submit”);
      b.addClickListener(new ButtonListener());
   }

   private void submit(){…}
}
```

Exercise

- Modify the inner class to only call submit the first time the button is clicked:

```
public class MyGui{
    //An inner class used as an adapter class
    private class ButtonListener implements ClickListener{

      public void onClick(ClickEvent evt){
               submit();
          }
      }
   }

   private void createSubmitButton(){
      Button b = new Button(“submit”);
      b.addClickListener(new ButtonListener());
   }

   private void submit(){…}
}
```

Java Inner Class Adapters

- Example:

```
public class MyGui{
    //An inner class used as an adapter class
    private class ButtonListener implements ClickListener{
      private boolean submitted_ = false;
      public void onClick(ClickEvent evt){
                if(!submitted_){
               submit();
               submitted_ = true;
          }
      }
   }

   private void createSubmitButton(){
      Button b = new Button(“submit”);
      b.addClickListener(new ButtonListener());
   }

   private void submit(){…}
}
```

Java Anonymous Inner Classes

- Problem: Sometimes you need to create a class that implements an interface in order to connect to another object.
- Example:

```
public class Button{
   private void addClickListener(ClickListener l){…}
}
```

```
public interface ClickListener{
   public void onClick(ClickEvent evt);
}
```

```
public class MyGui{
   private void createSubmitButton(){
      Button b = new Button();
      //how do I connect the button to my submit method
      //without requiring MyGui to implement ClickListener?
   }
   private void submit(){…}
}
```

Java Anonymous Inner Classes Adapters

- Solution: Java Anonymous Inner Classes
- An anonymous inner class is an unnamed class that is declared inline in a method
- Example:

```
public class MyGui{
   private void createSubmitButton(){
      Button b = new Button();

      //Anonymous inner class that creates a new implementation
      //of ClickListener in this method that serves as an adapter
      ClickListener l = new ClickListener(){
         public void onClick(ClickEvent evt){
           submit();
         }
      };
      b.setOnClickListener(l);
}
   private void submit(){…}
}
```

Exercise

- Solution: Java Anonymous Inner Classes
- An anonymous inner class is an unnamed class that is declared inline in a method
- Example:

```
public class MyGui{



  private Button createButton(SomeListener l){
      Button b = new Button();
      ButtonListener l = //your code

      //SomeListener has a “fired()” method that we would like
      //called when the ButtonListener’s “clicked()” method is called.
      //The ButtonListener defines a single “clicked()” method.
      //Define an inner class and instantiate an instance of it
      //to connect the ButtonListener to the fired() method of
      //SomeListener. You cannot use the final key word.
      return b;
  }
}
```

Exercise

- Solution: Java Anonymous Inner Classes
- An anonymous inner class is an unnamed class that is declared inline in a method
- Example:

```
public class MyGui{


  private Button createButton(SomeListener l){
      Button b = new Button();
      ButtonListener l = //your code

      //Connect clicked() to fired() as in the last example but use
      //an anonymous inner class instead
      return b;
  }
}
```