9.5/10 - Great job David. This shows that you are understanding the concepts in the class and applying them well. Test cases were very understandable. Please take a look at my comments. I hope that they help.

Thanks for making it easy to grade with the clear explanations. Again, great job.

David Pick
CM 2403

Very good!

Acceptance Tests:

1. When the application starts a window will be displayed
2. The window will have 3 buttons in the upper right corner for minimize, maximize, and close in that order
3. The window will have two drop down menus, Look & Feel and Help
4. The look and feel menu will allow the user to choose a theme for the application
5. The help menu will allow them to view an about page for the application
6. There will be a label displaying name of the application, the release date, and the build number
7. There will be four text boxes in the window
8. The first will display the timezone of the machine running the application
9. The second will display the current time of the machine running the application
10. The third will be empty
11. The fourth will also be empty
12. There will be a set clock button below the text boxes
13. When the set clock button is clicked the third text box will be filled in with the amount the machines clock needs to be changed, and the fourth text box will show the accurate time.
14. When the button is clicked the application will also set the machines clock to the correct time

Smell	Code	Line Number	
Magic Number	Thread.sleep(100);	351	Replace magic number with constant

Switch Statements	switch (stage)
	{
	case 0:
	// get Time from atomic clock
	getAtomicTime();
	break;
	
	case 1:
	// set both clocks to the same time.
	setTime();
	break;
	
	case 2:
	default:
	System.exit(0);
	}
		436	Replace conditional with State/Strategy

Magic Number	System.exit(0);	450	Replace magic number with constant

Long Method	if (stage > 0)	510	Replace temp with query

Duplicated Code	if (stage > 0)
	{
	if (correction < 0)
	{
	correctionLabel.setText("-Correction");
	correctionText.setText(inWords.toWords(-correction));
	}
	else
	{
	correctionLabel.setText("+Correction");
	correctionText.setText(inWords.toWords(correction));
	}
	}
	else
	{
	correctionLabel.setText("+Correction");
	correctionText.setText("");
	}
		530	Extract Method

Magic Number	final int minutes = (int) (millis / (1000 * 60));	577	Replace magic number with constant
Long Method	final String nearbySNTPServer = i % 3 + "." + country + ".pool.ntp.org";	621	Replace Temp with Query
Data Class	See Code	79 - 292	Move Method, Encapsulate Collection
Comments	See Code	1020 - 1085	Extract Method
Long Method	See Code	1020 - 1085	Extract method

Long Method	private void displayAccurateTime()
	{
	// intern to reduce garbage collection frequency
	// Avoid flicker.
	final String t;
	if (stage > 0)
	{
		505	Decompose Conditional

	FLOCAL.setTimeZone(
	TimeZone.getDefault());
	t = FLOCAL.format(
	accurateTime.getTime()
	).intern();
	}
	else
	{
	t = "";
	}
	if (!t.equals(
	accurateTimeText.getText()
	))
	{
	accurateTimeText.setText(
	t);
	}
	}// end
	displayAccurateTime		

While there were many more refactoring opportunities, I decided to limit my list to the first ten that I saw.

Refactorings:
Duplicated Code
Extract Method

Good!

This code is a good example of the duplicated code bad smell, because we see the same two lines being repeated with different parameters. They can easily be pulled out into a function to make the overall code simpler.

```
private void displayCorrection()
{
if ( stage > 0 )
{
if ( correction < 0 )
{
correctionLabel.setText( "- Correction" );
correctionText.setText( inWords.toWords( -correction ) );
}
else
{
correctionLabel.setText( "+ Correction" );
correctionText.setText( inWords.toWords( correction ) );
}
}
else
{
correctionLabel.setText( "+ Correction" );
correctionText.setText( "" );
}
}// end displayCorrection
```


```
private void displayCorrection()
{
if ( stage > 0 )
{
if ( correction < 0 )
{
setText(correctionLabel, correctionText, "- Correction", -1 * correction);
}
else
{
setText(correctionLabel, correctionText, "+Correction", correction);
}
}
else
{
setText(correctionLabel, correctionText, "+ Correction", correction);
}
}// end displayCorrection

private void setText(Label label, Textbox textBox, string textToSet, int correction) {
correctionLabel.setText( "- Correction" );
correctionText.setText( inWords.toWords( correction ) );
}
```

++
Long Method
Replace Temp with Query

Good!

IN this piece of code we see the temporary variable country created and then used once in the code. In this instance it makes the code clearer to simply make the call, rather than using a temporary variable.

```
long getPoolCorrection()
{
final String country = Locale.getDefault().getCountry().toLowerCase();
long total = 0;
for ( int i = 0; i < PROBES; i++ )
{
final String nearbySNTPServer = i % 3 + "." + country + ".pool.ntp.org";
System.out.println( "using timeserver " + nearbySNTPServer );

final long adjust = MiniSNTP.correction( nearbySNTPServer );
if ( adjust == Long.MAX_VALUE )
{
return Long.MAX_VALUE;
}
total += adjust;
}
return ( total + PROBES / 2 ) / PROBES;
}
```

```
long getPoolCorrection()
{
long total = 0;
for ( int i = 0; i < PROBES; i++ )
{
final String nearbySNTPServer = i % 3 + "." +
Locale.getDefault().getCountry().toLowerCase(); + ".pool.ntp.org";
System.out.println( "using timeserver " + nearbySNTPServer );

final long adjust = MiniSNTP.correction( nearbySNTPServer );
if ( adjust == Long.MAX_VALUE )
{
return Long.MAX_VALUE;
}
total += adjust;
}
return ( total + PROBES / 2 ) / PROBES;
}
```

+++

Long Method
Decompose Conditional

In this code we see a conditional that is overly complicated. We can completely remove the else statement and define t as the empty string in the beginning of the method.

Not certain your test for the conditional will work the same, but I trust you on this.

```
private void displayAccurateTime()
{
// intern to reduce garbage collection frequency
// Avoid flicker.
final String t;
if ( stage > 0 )
{
FLOCAL.setTimeZone( TimeZone.getDefault() );
t = FLOCAL.format( accurateTime.getTime() ).intern();
}
else
{
t = "";
}
if ( !t.equals( accurateTimeText.getText() ) )
{
accurateTimeText.setText( t );
}
}// end displayAccurateTime
```

```
private void displayAccurateTime()
{
// intern to reduce garbage collection frequency
// Avoid flicker.
final String t;
t = "";
if ( stage > 0 )
{
FLOCAL.setTimeZone( TimeZone.getDefault() );
t = FLOCAL.format( accurateTime.getTime() ).intern();
}

accurateTimeText.setText( t );

}// end displayAccurateTime
```

++
Switch Statements
Recplace Type Code with State/Strategy

In this code we see a switch statement, that is switching off of an integer. This makes the code hard to follow as it is difficult to understand what each case of the switch is doing. By creating abstract classes with the type codes in them we can switch on those type codes making the code much easier to understand.

```
switch ( stage )
{
case 0:
// get Time from atomic clock
getAtomicTime();
break;

case 1:
// set both clocks to the same time.
setTime();
break;

case 2:
default:
System.exit( 0 );
}
```

Good...

```
switch ( getType() )
{
case Time.ATOMICTIME:
// get Time from atomic clock
getAtomicTime();
break;

case Time.SETTIME:
```

```
// set both clocks to the same time.
setTime();
break;

case Time.EXIT:
default:
System.exit( 0 );
}

abstract class Time {
    abstract int getTypeCode();
}

class AtomicTime extends Time {
    int getTypeCode () {
        return Time.ATOMICTIME
    }
}

class SetTime extends Time {
    int getTypeCode () {
        return Time.SETTIME
    }
}

class Exit extends Time {
    int getTypeCode () {
        return Time.EXIT
    }
}
```