Running the Basic Contexts and Dependency Injection Examples

This chapter describes in detail how to build and run simple examples that use CDI.

The examples are in the following directory:

tut-install`/examples/cdi/`

To build and run the examples, you will do the following:

1. Use NetBeans IDE or the Ant tool to compile and package the example.

2. Use NetBeans IDE or the Ant tool to deploy the example.

3. Run the example in a web browser.

Each example has a `build.xml` file that refers to files in the following directory:

tut-install`/examples/bp-project/`

See Chapter 2, Using the Tutorial Examples, for basic information on installing, building, and running the examples.

The following topics are addressed here:

- The `simplegreeting` CDI Example
- The `guessnumber` CDI Example

The `simplegreeting` CDI Example

The `simplegreeting` example illustrates some of the most basic features of CDI:

scopes, qualifiers, bean injection, and accessing a managed bean in a JavaServer Faces application.

When you run the example, you click a button that presents either a formal or an informal greeting, depending on how you edited one of the classes.

The example includes four source files, a Facelets page and template, and configuration files.

The `simplegreeting` Source Files

The four source files for the `simplegreeting` example are

- The default `Greeting` class, shown in Beans as Injectable Objects

- The `@Informal` qualifier interface definition and the `InformalGreeting` class that implements the interface, both shown in Using Qualifiers

- The `Printer` managed bean class, which injects one of the two interfaces, shown in full in Adding Setter and Getter Methods

The source files are located in the following directory:

tut-install`/examples/cdi/simplegreeting/src/java/greetings`

The Facelets Template and Page

To use the managed bean in a simple Facelets application, you can use a very simple template file and `index.xhtml` page.

The template page, `template.xhtml`, looks like this:

```
<?xml version='1.0'
encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html
xmlns=
"http://www.w3.org/1999/xhtml"
xmlns:h=
"http://java.sun.com/jsf/html"
```

```
xmlns:ui=
"http://java.sun.com/jsf/facelets">
<h:head>
<meta http-equiv="Content-Type"
content=
"text/html; charset=UTF-8"
/>
<link
href="resources/css/default.css"
rel="stylesheet" type="text/css"
/>
```

```
<title>
<ui:insert name="title">
Default Title
</ui:insert>
</title>
</h:head>
<body>
<div id="container">
<div id="header">
<h2>
```

```
<ui:insert name="head">
Head
</ui:insert></h2>
</div>
<div id="space">
<p></p>
</div>
<div id="content">
<ui:insert name="content"/>
</div>
</div></body></html>
```

To create the Facelets page, you can redefine the title and head, then add a small form to the content:

```
<?xml version='1.0'
encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xh
tml1-transitional.dtd">
```

```
<html
xmlns=
"http://www.w3.org/1999/xhtml"
xmlns:ui=
"http://java.sun.com/jsf/facelets"
xmlns:h=
"http://java.sun.com/jsf/html"
xmlns:f=
"http://java.sun.com/jsf/core">
<ui:composition
template="/template.xhtml">
```

```
<ui:define name="title">
Simple Greeting
</ui:define>
<ui:define name="head">
Simple Greeting
</ui:define>
<ui:define name="content">
<h:form id="greetme">
<p><h:outputLabel
value="Enter your name: "
for="name"/>
```

```
<h:inputText id="name"
value="#{printer.name}"/></p>
<p>
<h:commandButton value="Say Hello"
action=
"#{printer.createSalutation}"
/></p>
<p>
<h:outputText
value="#{printer.salutation}"/>
</p>
```

```
</h:form>
</ui:define>
</ui:composition>
</html>
```

The form asks the user to type a name.

The button is labeled Say Hello, and the action defined for it is to call the `createSalutation` method of the `Printer` managed bean.

This method in turn calls the `greet` method of the defined `Greeting` class.

The output text for the form is the value of the greeting returned by the setter method.

Depending on whether the default or the `@Informal` version of the greeting is injected, this is one of the following, where *name* is the name typed by the user:

Hello, *name*.
Hi, *name*!

The Facelets page and template are located in the following directory:

tut-install`/examples/cdi/simplegreeting/web`

The simple CSS file that is used by the Facelets page is in the following location:

tut-install`/examples/cdi/simplegreeting/web/resources/css/default.css`

Configuration Files

You must create an empty `beans.xml` file to indicate to GlassFish Server that your application is a CDI application.

This file can have content in some situations, but not in simple applications like this one.

Your application also needs the basic web application deployment descriptors `web.xml` and `glassfish-web.xml`.

These configuration files are located in the following directory:

tut-install`/examples/cdi/simplegreeting/web/WEB-INF`

Building, Packaging, Deploying, and Running the `simplegreeting` CDI Example

You can build, package, deploy, and run the `simplegreeting` application by using either NetBeans IDE or the Ant tool.

To Build, Package, and Deploy the `simplegreeting` Example Using NetBeans IDE

This procedure builds the application into the following directory:

***tut-install*`/examples/cdi/simplegreeting/build/web`**

The contents of this directory are deployed to the GlassFish Server.

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:
   `tut-install/examples/cdi/`

3. Select the `simplegreeting` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

6. (Optional) To modify the `Printer.java` file, perform these steps:

   a. Expand the Source Packages node.
   b. Expand the `greetings` node.

c. Double-click the `Printer.java` file.

d. In the edit pane, comment out the `@Informal` annotation:

e. `//@Informal`

f.
```
@Inject
Greeting greeting;
```

g. Save the file.

7. In the Projects tab, right-click the `simplegreeting` project and select Deploy.

To Build, Package, and Deploy the `simplegreeting` Example Using Ant

1. In a terminal window, go to:

   `tut-install/examples/cdi/simplegreeting/`

2. Type the following command:

   `ant`

This command calls the `default` target, which builds and packages the application into a WAR file, `simplegreeting.war`, located in the `dist` directory.

3. Type the following command:

```
ant deploy
```

Typing this command deploys `simplegreeting.war` to the GlassFish Server.

To Run the `simplegreeting` Example

1. In a web browser, type the following URL:

`http://localhost:8080/simplegreeting`

The Simple Greeting page opens.

2. Type a name in the text field.

For example, suppose that you type `Duke`.

3. Click the Say Hello button.

If you did not modify the `Printer.java` file, the following text string appears below the button:

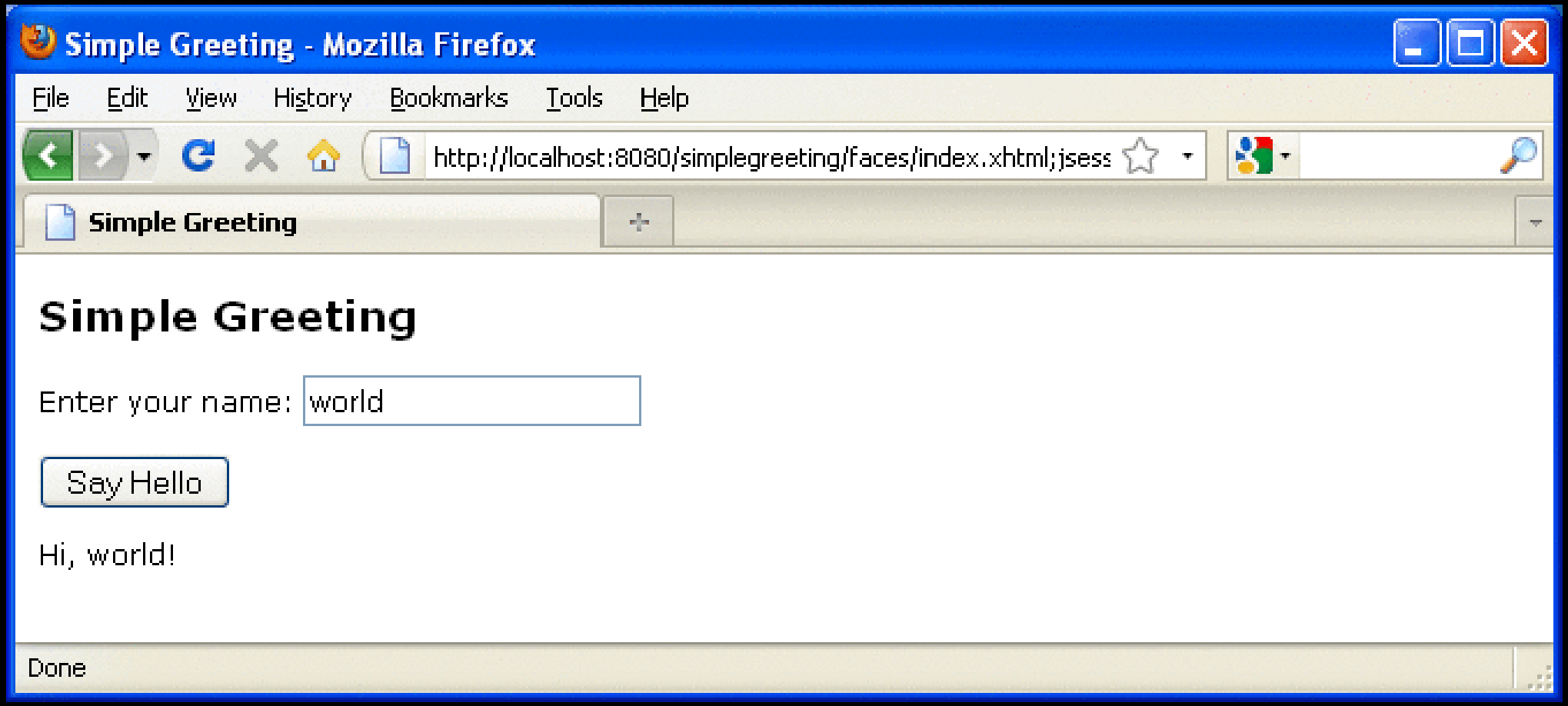
```
Hi, Duke!
```

If you commented out the `@Informal` annotation in the `Printer.java` file, the following text string appears below the button:

```
Hello, Duke.
```

Figure 29-1 shows what the application looks like if you did not modify the `Printer.java` file.

Figure 29-1 Simple Greeting Application



The `guessnumber` CDI Example

The `guessnumber` example, somewhat more complex than the `simplegreeting` example, illustrates the use of producer methods and of session and application scope.

The example is a game in which you try to guess a number in fewer than ten attempts.

It is similar to the `guessnumber` example described in Chapter 5, Introduction to Facelets, except that you can keep guessing until you get the right answer or until you use up your ten attempts.

The example includes four source files, a Facelets page and template, and configuration files.

The configuration files and the template are the same as those used for the `simplegreeting` example.

The `guessnumber` Source Files

The four source files for the `guessnumber` example are

- The `@MaxNumber` qualifier interface
- The `@Random` qualifier interface
- The `Generator` managed bean, which defines producer methods
- The `UserNumberBean` managed bean

The source files are located in the following directory:

tut-install`/examples/cdi/guessnumber/src/java/guessnumber`

The @MaxNumber and @Random Qualifier Interfaces

The @MaxNumber qualifier interface is defined as follows:

```
package guessnumber;
import static java.lang.annotation.
ElementType.FIELD;
```

```
import static java.lang.annotation.
ElementType.METHOD;
import static java.lang.annotation.
ElementType.PARAMETER;
import static java.lang.annotation.
ElementType.TYPE;
import static java.lang.annotation.
RetentionPolicy.RUNTIME;
import
java.lang.annotation.Documented;
```

```
import
java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.inject.Qualifier;
@Target
({ TYPE, METHOD, PARAMETER, FIELD})
@Retention(RUNTIME)
@Documented
@Qualifier
public @interface MaxNumber {
}
```

The @Random qualifier interface is defined as follows:

```
package guessnumber;
import static java.lang.annotation.
ElementType.FIELD;
import static java.lang.annotation.
ElementType.METHOD;
import static java.lang.annotation.
ElementType.PARAMETER;
```

```
import static java.lang.annotation.
ElementType.TYPE;
import static java.lang.annotation.
RetentionPolicy.RUNTIME;
import
java.lang.annotation.Documented;
import
java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.inject.Qualifier;
```

```
@Target
({ TYPE, METHOD, PARAMETER, FIELD})
@Retention(RUNTIME)
@Documented
@Qualifier
public @interface Random {}
```

The `Generator` Managed Bean

The `Generator` managed bean contains the two producer methods for the application.

The bean has the `@ApplicationScoped` annotation to specify that its context extends for the duration of the user's interaction with the application:

```
package guessnumber;
import java.io.Serializable;
import javax.enterprise.context.
ApplicationScoped;
import
javax.enterprise.inject.Produces;
@ApplicationScoped
public class Generator implements
Serializable {
```

```
private static final
long serialVersionUID = -
7213673465118041882L;
private java.util.Random random =
new java.util.Random
( System.currentTimeMillis() );
private int maxNumber = 100;
java.util.Random getRandom()
{ return random; }
```

```
@Produces @Random int next() {
return
getRandom().nextInt(maxNumber);
}
@Produces
@MaxNumber int getMaxNumber()
{ return maxNumber; }
}
```

The `UserNumberBean` Managed Bean

The `UserNumberBean` managed bean, the backing bean for the JavaServer Faces application, provides the basic logic for the game.

This bean does the following:

- Implements setter and getter methods for the bean fields
- Injects the two qualifier objects
- Provides a `reset` method that allows you to begin a new game after you complete one
- Provides a `check` method that determines whether the user has guessed the number

- Provides a `validateNumberRange` method that determines whether the user's input is correct

The bean is defined as follows:

```
package guessnumber;
import java.io.Serializable;
import
javax.annotation.PostConstruct;
```

```
import javax.enterprise.context.
SessionScoped;
import
javax.enterprise.inject.Instance;
import javax.inject.Inject;
import javax.inject.Named;
import javax.faces.application.
FacesMessage;
import
javax.faces.component.UIComponent;
```

```
import
javax.faces.component.UIInput;
import
javax.faces.context.FacesContext;
@Named
@SessionScoped
public class UserNumberBean
implements Serializable {
private static final
long serialVersionUID = 1L;
private int number;
```

```
private Integer userNumber;
private int minimum;
private int remainingGuesses;
@MaxNumber
@Inject
private int maxNumber;
private int maximum;
@Random
@Inject
Instance<Integer> randomInt;
public UserNumberBean() { }
```

```
public int getNumber()
{ return number; }
public void setUserNumber
(Integer user_number)
{ userNumber = user_number; }
public Integer getUserNumber()
{ return userNumber; }
public int getMaximum()
{ return (this.maximum); }
public void setMaximum(int maximum)
{ this.maximum = maximum; }
```

```
public int getMinimum()
{ return (this.minimum); }
public void setMinimum(int minimum)
{ this.minimum = minimum; }
public int getRemainingGuesses()
{ return remainingGuesses; }
public String check()
throws InterruptedException {
if (userNumber > number)
{ maximum = userNumber - 1; }
if (userNumber < number)
```

```
{ minimum = userNumber + 1; }
if (userNumber == number) {
FacesContext.getCurrentInstance().
addMessage
(null,new FacesMessage("Correct!"));
}
remainingGuesses--;
return null;
}
```

```
@PostConstruct
public void reset() {
this.minimum = 0;
this.userNumber = 0;
this.remainingGuesses = 10;
this.maximum = maxNumber;
this.number = randomInt.get();
}
public void validateNumberRange
(FacesContext context,
UIComponent toValidate,Object value){
```

```
if (remainingGuesses <= 0) {
FacesMessage message =
new FacesMessage("No guesses left!");
context.addMessage(
toValidate.getClientId(context),
message);
((UIInput)toValidate).setValid(false);
return;
}
int input = (Integer) value;
```

```
if
(input <minimum || input>maximum){
((UIInput) toValidate).
setValid(false);
FacesMessage message =
new FacesMessage("Invalid guess");
context.addMessage(
toValidate.getClientId(context),
message);
}
}}
```

The Facelets Page

This example uses the same template that the `simplegreeting` example uses.

The `index.xhtml` file, however, is more complex.

```
<?xml version='1.0'
encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html
xmlns=
"http://www.w3.org/1999/xhtml"
xmlns:ui=
"http://java.sun.com/jsf/facelets"
```

```
xmlns:h=
"http://java.sun.com/jsf/html"
xmlns:f=
"http://java.sun.com/jsf/core"
>
<ui:composition
template="/template.xhtml">
<ui:define name="title">
Guess My Number
</ui:define>
```

```
<ui:define name="head">
Guess My Number
</ui:define>
<ui:define name="content">
<h:form id="GuessMain">
<div
style="
color: black;
font-size: 24px;
">
```

```
<p>
I'm thinking of a number between
<span style="color: blue">
#{userNumberBean.minimum}
</span> and
<span style="color: blue">
#{userNumberBean.maximum}
</span>. You have
<span style="color: blue">
#{userNumberBean.remainingGuesses}
</span> guesses.</p>
```

```
</div>
<h:panelGrid border="0" columns="5"
style="font-size: 18px;">
Number:
<h:inputText id="inputGuess"
value=
"#{userNumberBean.userNumber}"
required="true" size="3"
disabled=
"#{userNumberBean.number eq
userNumberBean.userNumber}"
```

```
validator=
"#{userNumberBean.validateNumberRange}"
>
</h:inputText>
<h:commandButton id="GuessButton"
value="Guess"
action="#{userNumberBean.check}"
disabled="#{userNumberBean.number
eq userNumberBean.userNumber}"
/>
```

```
<h:commandButton id="RestartButton"
value="Reset"
action="#{userNumberBean.reset}"
immediate="true"
/>
<h:outputText id="Higher"
value="Higher!"
rendered="#{userNumberBean.number
gt userNumberBean.userNumber and
userNumberBean.userNumber ne 0}"
style="color: red"/>
```

```
<h:outputText id="Lower"
value="Lower!"
rendered="#{userNumberBean.number
lt userNumberBean.userNumber and
userNumberBean.userNumber ne 0}"
style="color: red"/>
</h:panelGrid>
<div
style="color: red;
font-size: 14px;">
```

```
<h:messages id="messages"
globalOnly="false"
/>
</div>
</h:form>
</ui:define>
</ui:composition>
</html>
```

The Facelets page presents the user with the minimum and maximum values and the number of guesses remaining.

The user's interaction with the game takes place within the `panelGrid` table, which contains an input field, Guess and Reset buttons, and a text field that appears if the guess is higher or lower than the correct number.

Every time the user clicks the Guess button, the `userNumberBean.check` method is called to reset the maximum or minimum value or, if the guess is correct, to generate a `FacesMessage` to that effect.

The method that determines whether each guess is valid is `userNumberBean.validateNumberRange`.

Building, Packaging, Deploying, and Running the `guessnumber` CDI Example

You can build, package, deploy, and run the `guessnumber` application by using either NetBeans IDE or the Ant tool.

To Build, Package, and Deploy the `guessnumber` *Example Using NetBeans IDE*

This procedure builds the application into the following directory:

`tut-install/examples/cdi/guessnumber/build/web`

The contents of this directory are deployed to the GlassFish Server.

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:
   `tut-install/examples/cdi/`

3. Select the `guessnumber` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

6. In the Projects tab, right-click the `guessnumber` project and select Deploy.

To Build, Package, and Deploy the `guessnumber` Example Using Ant

1. In a terminal window, go to:

   `tut-install/examples/cdi/guessnumber/`

2. Type the following command:

   `ant`

This command calls the `default` target, which builds and packages the application into a WAR file, `guessnumber.war`, located in the `dist` directory.

3. Type the following command:

```
ant deploy
```

The `guessnumber.war` file will be deployed to the GlassFish Server.

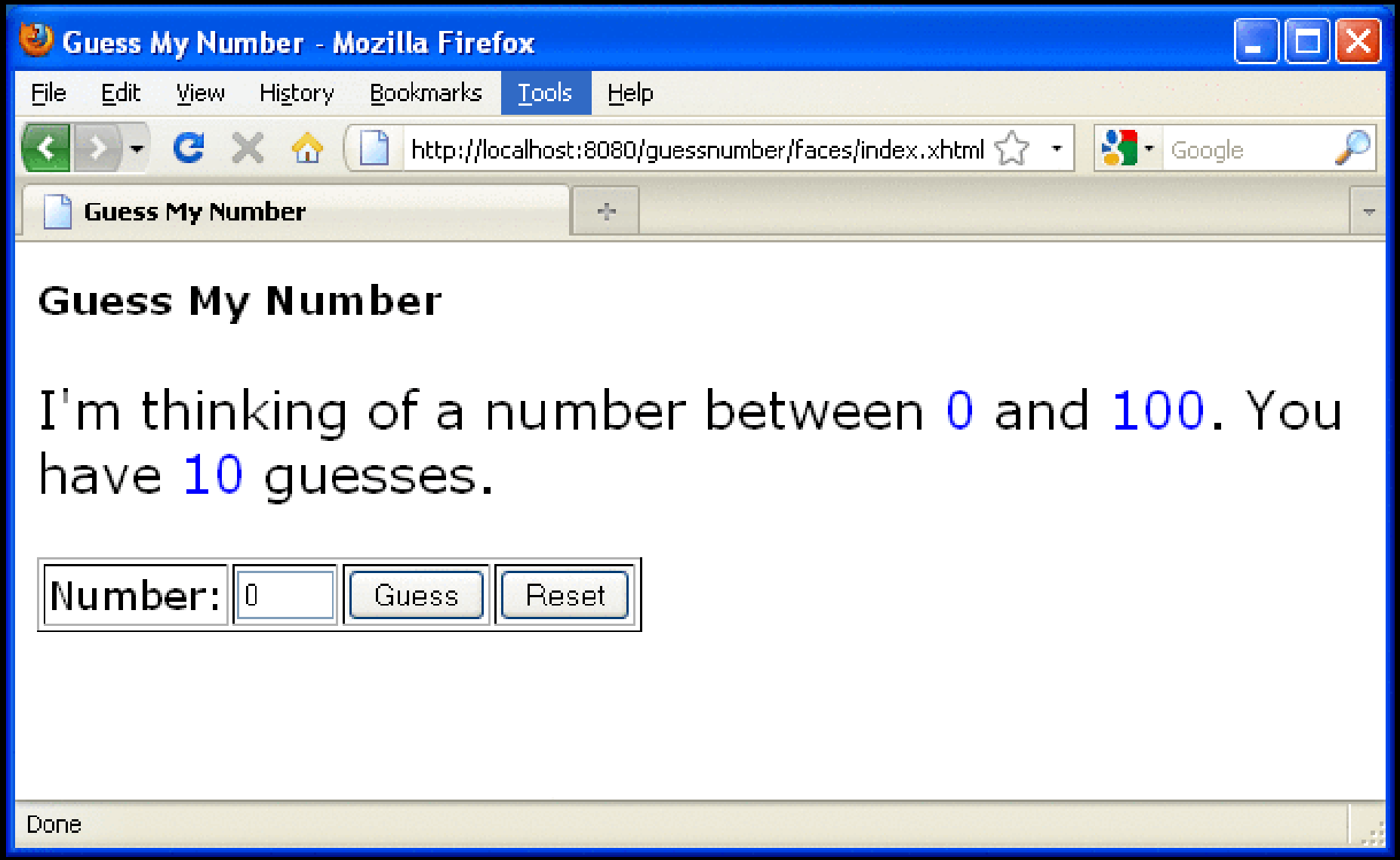
To Run the `guessnumber` Example

1. In a web browser, type the following URL:

```
http://localhost:8080/guessnumber
```

The Guess My Number page opens, as shown in Figure 29-2.

Figure 29-2 Guess My Number Example



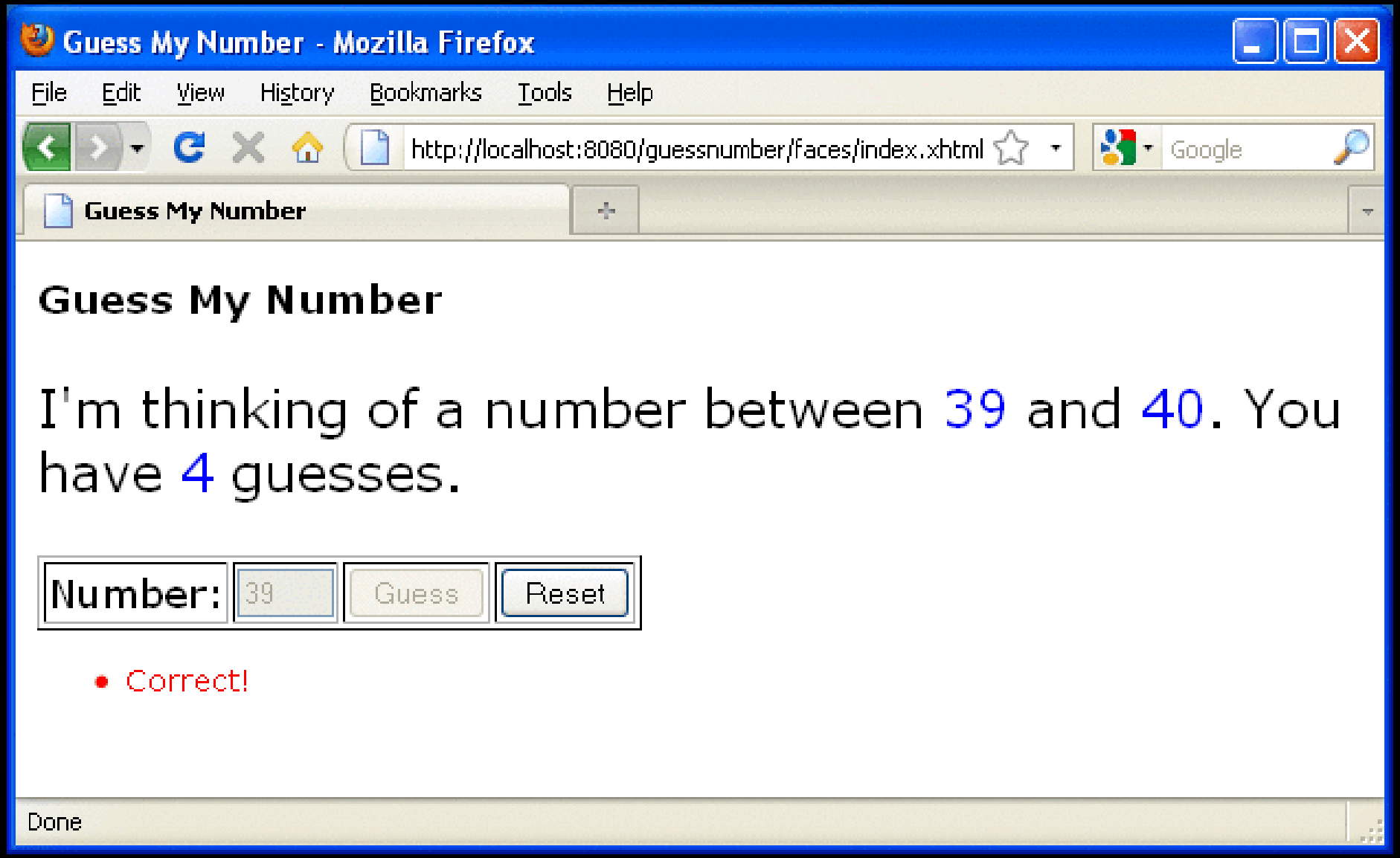
2. Type a number in the Number text field and click Guess.

The minimum and maximum values are modified, along with the remaining number of guesses.

3. Keep guessing numbers until you get the right answer or run out of guesses.

If you get the right answer, the input field and Guess button are grayed out, as shown in Figure 29-3.

Figure 29-3 Guess My Number at End of Game



4. Click the Reset button to play the game again with a new random number.