Practices for Lesson 12

Practices Overview

In these practices, you experiment with handling checked exceptions. In the first practice, you handle an exception thrown by one of the Java foundation classes. In the second practice, you catch and throw a custom exception class.

Practice 12-1: Using a `try/catch` Block to Handle an Exception

Overview

In this practice, you use the Java API documentation to examine the SimpleDateFormat class and find the exception thrown by its `parse` method. Then you create a class that calls the `parse` method, using a `try/catch` block to handle the exception.

Assumptions

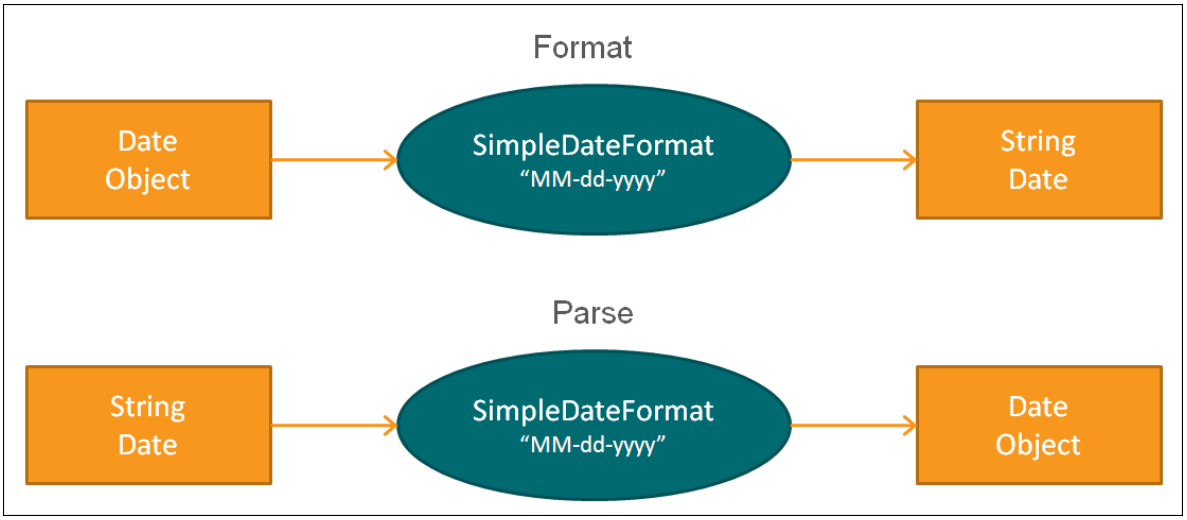
This practice assumes that the following file appears in the practice folder for this lesson, `Lesson12`:

- DateTest.java

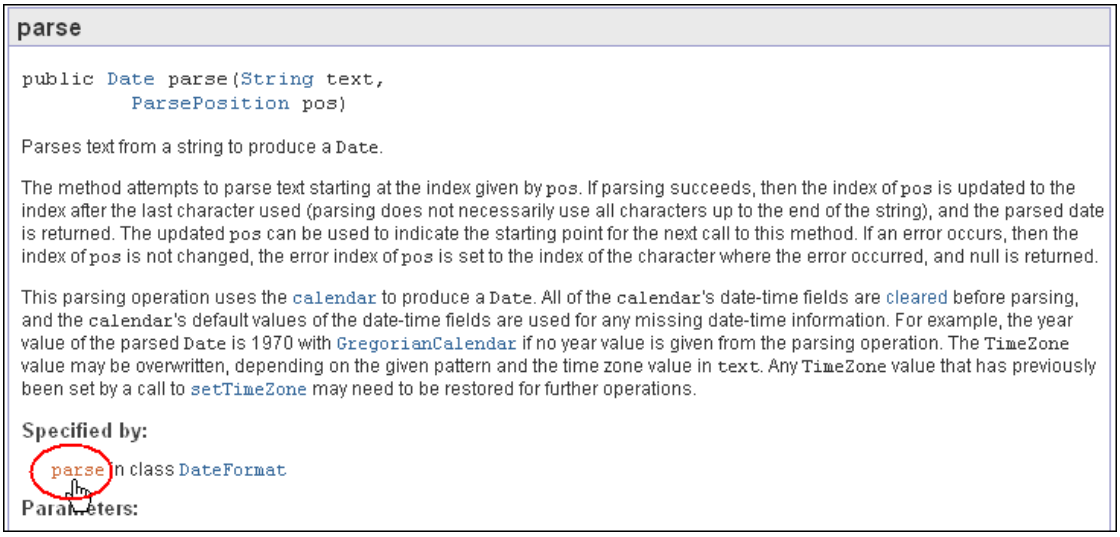
Tasks

1. In NetBeans, create a new project from existing sources called Practice13. Set the Source Package Folder to point to `Lesson12`. Remember to set the Binary Source Format property of the project. If you need additional details, refer to Practice 1-2, Steps 3 and 4.

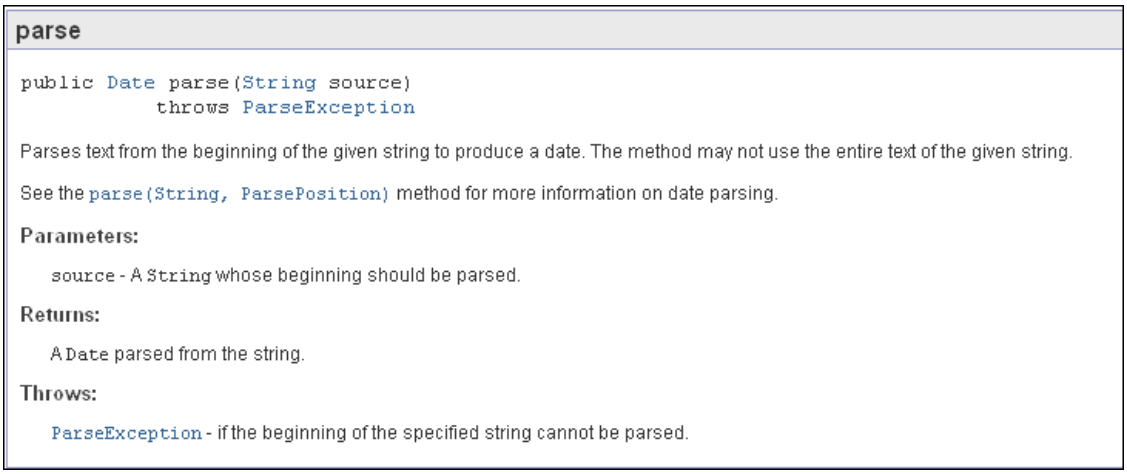
   There are many files in this project. Only the DateTest class is relevant for this practice.
2. Open the Java API Specification documentation by using the shortcut on the desktop.
3. Find the SimpleDateFormat class in the `java.text` package. This class allows you to pick a standard date format that will then be applied during both formatting and parsing. For instance, you *format* the String output of a Date object, and you *parse* (or create) a Date object based on a formatted String representation of the date.



4. The steps below will guide your examination of the SimpleDateFormat documentation.
   a. Find and click the `parse` method. As you can see, this method has two arguments. In this practice, you invoke a simpler `parse` method that belongs to the superclass, DateFormat, instead of this `parse` method you see here. The superclass method is not `private` and is therefore, available to a SimpleDateFormat object.
   b. In the **Specified by** section, click the `parse` link as shown below to go to the DateFormat documentation for this method.

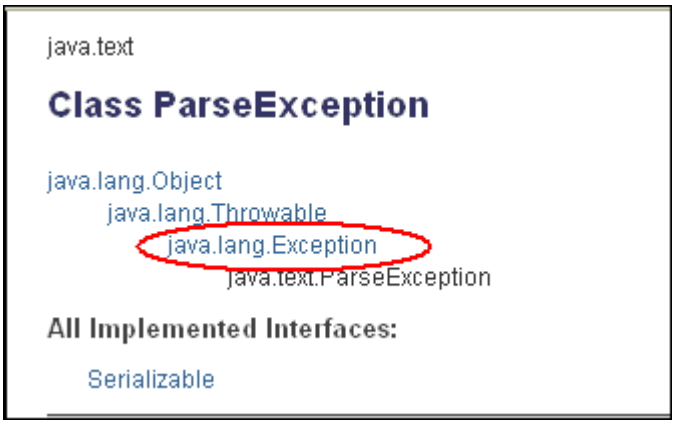


c. The javadocs now display a similar two-argument `parse` method in the DateFormat class. Scroll up to the one-argument `parse` method directly above this one.



d. Notice that this `parse` method accepts a single `String` argument and returns a `Date` object. What, if any, exceptions does it throw?

e. Is the ParseException a checked exception (one that must be caught or thrown)? Click the ParseException link to see its class hierarchy. Is it a subclass of Exception? If so, it is a checked exception and must be handled in the program.



f. Click the browser's Back button twice to return to the SimpleDateFormat documentation.

g. Find the `format` method. This `format` method accepts three arguments. Once again, there is a simpler version of this method defined in the DateFormat class that you use in this practice. Click the `format` link under the **Specified by** heading to view the DateFormat documentation for this method.

h. Scroll down to find the one-argument `format` method. Notice that it accepts a single `Date` object argument and returns a `String` value. Does this `format` method throw any exceptions? (*Answer:* It does not.)

5. In NetBeans, create a new Java Class called "DateManipulator". Provide field declarations as indicated in the table below. More detailed instructions follow the table.

Step	Code Description	Choices or Values
a.	Declare a field of type `Date`	Name: `myDate`
b.	Declare and instantiate a field of type `SimpleDateFormat` specifying its default format in the constructor.	Name: `simpleDF` Default format: `"MM/dd/yyyy"`
c.	Add the necessary import statements	`java.util.Date` `java.text.SimpleDateFormat`

a. Declare a field of type `Date`, using the variable name `myDate`.

b. Declare a field of type `SimpleDateFormat` called `simpleDF`. Use the `new` operator to instantiate (create an object) the SimpleDateFormat field in the same line as the declaration. Specify the default format for this object by passing the format String to the object constructor as shown below.

```
SimpleDateFormat simpleDF =new SimpleDateFormat("MM/dd/yyyy");
```

c. Click the error icons that appear next to each of these lines of code. Select the option to add the required `import` statements. There are two possible Date objects that can be imported. Choose the `java.util.Date`.

6. Add a public method called `parseDate` that accepts a `String` argument called `dateString` and returns `void`.

Note: This method creates a Date object instance by invoking the `parse` method. It formats the Date object according to the default format of the SimpleDateFormat object and displays the resulting string. It also displays the native date format of the Date object for comparison. In addition to this, the method handles the ParseException using a `try / catch` block.

7. Follow the high-level steps in the table below to code the `parseDate` method. More detailed steps are provided following the table.

Step	Code Description	Choices or Values
a.	Declare a local `String` variable.	Name: `formattedDateString`
b.	Invoke the `parse` method of the SimpleDateFormat object. Ignore the error sign for now.	Pass the `dateString` as the `String` argument. Assign the return value to the `myDate` field.
c.	Display a message indicating that the `parse` method was successful	

Step	Code Description	Choices or Values
d.	Display the natively formatted `date` object	Print the Date object, itself, with a suitable label.
e.	Invoke the `format` method of the SimpleDateFormat object, passing `myDate` as the method argument.	Assign the return value to `formattedDateString`
f.	Display the formatted date String with a suitable label	

g.	Enclose all of the above code in a `try` block	`try{`
		`  // lines of code here`
		`}`

h.	Catch the ParseException and display the exception object	`catch (ParseException ex) {`
		`  // display the ex object here`
		`}`

i.	Add the missing import statement	`java.text.ParseException`

a. Declare a local `String` variable called `formattedDateString`. This will be used to hold the String representation of the formatted Date object.

b. Invoke the `parse` method of the `simpleDF` object, passing the method's `dateString` argument to the `parse` method. This method returns a Date object so assign the return value to `myDate`. You will, no doubt, notice an error icon in the left margin for this line of code. Put your cursor over it to see the warning message. You will add a try/catch block later to this fix this.

c. Use `System.out.println` to print the message "Parse successful".

d. Again use `System.out.println` to print `myDate` along with the message "Date with native format: ". (Hint: the Java Virtual Machine will invoke the `toString` method of the Date object.)

e. Invoke the `format` method of the `simpleDF` object. Pass `myDate` as the argument to the method. Assign the return value to `formattedDateString`.

```
formattedDateString = simpleDF.format(myDate);
```

f. Display `formattedDateString` with a suitable label. Suggestion "Formatted date: " + `formattedDateString`

g. Now you fix the error regarding the missing `try/catch` block. Surround all of the above lines of code in a `try` block.

- Hint: Right-click anywhere in the editor and select **Format** to correct the indentation of your code.

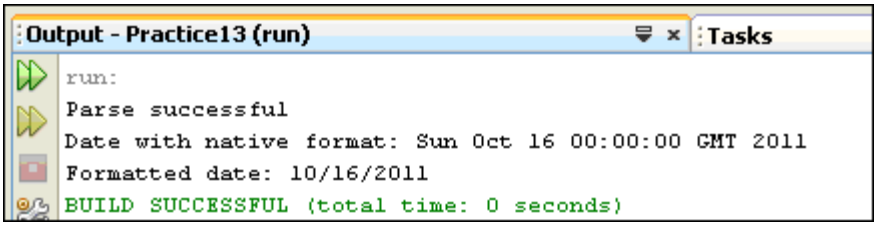
h. On the next line after the closing brace of the `try` block, add a `catch` block that catches the ParseException and displays the exception object to the screen.

i. Right-click the error icon in the left margin and allow NetBeans to provide the missing import statement (java.text.ParseException).

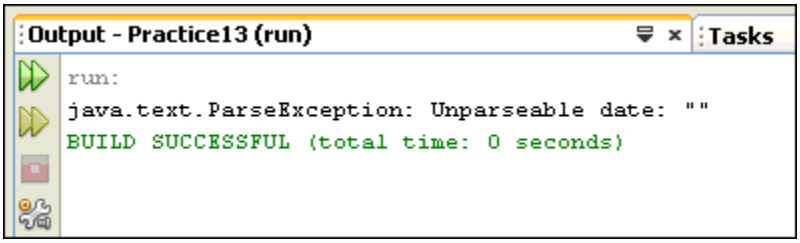
Solution:

```
public void parseDate(String myDate){
     try{
        String formattedDateString;
        myDate = simpleDF.parse(dateString);
        System.out.println("Parse successful");
        System.out.println("Date with native format: "
                + myDate);
        formattedDateString = simpleDF.format(myDate);
        System.out.println("Formatted date: "
              + formattedDateString);
     }catch (ParseException ex) {
         System.out.println(ex);
     }
}
```

8. Save and compile your program.
9. Open the DateTest class and examine it. Substitute your own date between the quotation marks in the `parseDate` method call. Use the format, “MM/dd/yyyy”, as specified in the comment.
10. Click Save to compile.
11. Run the DateTest and check the output. If your date was formatted correctly, the ParseException will not appear in the output. You should, however, see the difference between the native Date formatting and the effect of the SimpleDateFormat class on the formatting of that same date.



12. Now change the argument value of the `parseDate` method in DateTest so it reverts to being an empty string (“”). Save and compile the program.
13. Run the DateTest class again. The ParseException will be thrown this time and you should see the message from the exception object in the output.



Note: You will notice that the “Parse successful” message does not appear. That particular display occurred in the line immediately following the `parse` method call. When the `parse` method threw the exception, the program went immediately to the `catch` block and the remaining lines of code in the `try` block were not executed.

Practice 12-2: Catching and Throwing a Custom Exception

Overview

In this practice, you use a custom exception called “InvalidSkillException”. You use this with the Employee Tracking application that you designed and built in Practices 12-1 and 12-2. You throw the InvalidSkillException in one method and catch it in the calling method.

A new set of Java source files for the Employee hierarchy are provided for your use in this practice.

Assumptions

This practice assumes that the following files appear in the practice folder for this lesson and, consequently, in the Practice13 project:

- Editor.java
- Employee.java
- EmployeeTest.java
- GraphicIllustrator.java
- InvalidSkillException.java
- Manager.java
- Printable.java
- Printer.java
- SkilledEmployee.java
- TechnicalWriter.java

Tasks

Assume that there is a list of valid skill types that can be associated with a particular job role in the Employee Tracking system. In the `setSkill` method (belonging to the SkilledEmployee class), some validation is necessary to determine whether the skill argument passed into the method is valid for the employee’s job title. If the skill is not valid, the method will throw an InvalidSkillException. The calling method in the EmployeeTest class must then catch this exception.

Note: In our simple example, the validation in the `setSkill` method will be greatly simplified and does not represent the robust, thorough type of validation that would occur in a “real world” application. Our purpose here is to focus on catching and throwing exceptions.

1. Open the InvalidSkillException class and examine the code. It is very simple. The only thing that makes this function an exception is that it extends Exception. You see a public no-args constructor and also a public constructor that accepts a String argument. That argument is the message that will be displayed when this Exception object is printed.
2. Open the SkilledEmployee class and modify the `setSkill` method as described in the steps below. The solution for the `setSkill` method is shown following the steps if you need further assistance.
   a. Add a `throws` clause to the method declaration so that it `throws` an InvalidSkillException.
   b. As the first line of code in the method, declare a `boolean` variable called “valid” and initialize it to `true`

c. Use an `if/else` construct to set the value of the `valid` variable to `false` if the `skill` argument is `null` or has a length of less than 5. Otherwise, set `valid = true`.

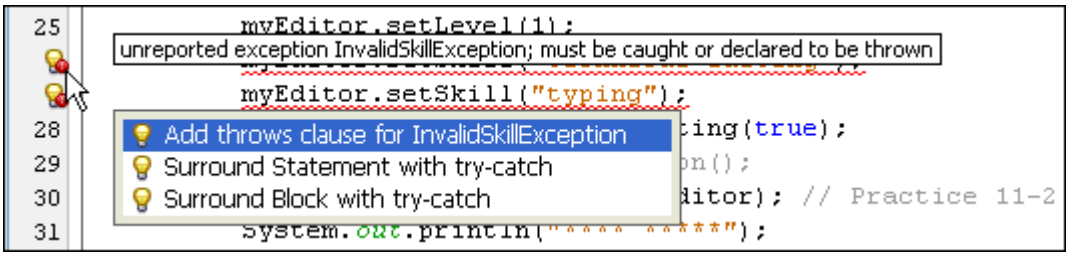
d. Use another `if/else` construct to test the value of the `valid` variable.

- If it is `true`, add the skill to the `skillList`.
- If it is `false`, throw a `new InvalidSkillException`, using the constructor that takes a `String` argument for the exception message.
- The message should show the `skill` argument that caused the exception and concatenate that to a string literal that indicates that this is an invalid value for an employee with this particular job. Also display the employee's job title, using `this.getJobTitle()`.

Solution:

```
public void setSkill(String skill) throws
     InvalidSkillException {
     Boolean valid = true;
     if(skill == null | skill.length() < 5){
        valid = false;
     }
     else {
        valid = true;
     }
     if (!valid) {
        throw new InvalidSkillException(skill +
          " is not valid for the " +
          this.getJobTitle() + " job.");
     }
     else {
        skillList.add(skill);
     }
}
```

3. Save and compile your program
4. Open the EmployeeTest class. You should now see red error icons in the left margin for every line of code that calls the `setSkill` method. Click one of the error icons to read the error description and see the options it offers to help you solve the problem.



When you compiled the SkilledEmployee class, you made NetBeans aware of the fact that the `setSkill` method throws an InvalidSkillException. The compiler is checking this (remember, this is a "checked exception") and expects you to either catch it or re-throw it

whenever you invoke `setSkill`. None of the suggested options will work well in this particular situation, so you add the `try/catch` block yourself.

5. Surround each *set* of `setSkill` method invocations with a `try/catch` block. In the `catch` block, display the exception object. You will have to add `try/catch` blocks for `myEditor`, `myGI`, and `myTW`.

   Example: The two invocations for the `myEditor` object can all go within a single block.

```
try{
     myEditor.setSkill("typing");
     myEditor.setSkill("technical editing");
}
catch(InvalidSkillException ex){
     System.out.println(ex);
}
```

6. Change the String argument in one of the `setSkill` calls to a shortened String (less than 5 characters) so that it will be deemed invalid and the exception will be thrown.
7. Save and compile your program. Run the EmployeeTest class and examine the output.

```
**** *****
InvalidSkillException: tw is not a valid skill for the Technical Writer job.
Name: James Ralph
Job Title:Technical Writer
Employee ID: 3
Level: 1
**** *****
Name: Susan Smith
Job Title:Manager
Employee ID: 4
Level: 2
Manager has the following employees:
        Fred Hanson
        Frank Moses
        James Ralph
BUILD SUCCESSFUL (total time: 0 seconds)
```

Note: The catching of an InvalidSkillException did not prevent the remainder of the method from executing. It prevented only the saving of an invalid skill for this employee. Handling checked exceptions in this way also offers an opportunity to write the error information to a log file or to prompt the user to enter the skill again (assuming a different user interface than we are using here).