Chapter 5

How to code methods, handle exceptions, and validate data

Objectives

Applied

1. Given an application that uses the console to get input from the user, write code that handles any exceptions that might occur.
2. Given an application that uses the console to get input from the user and the validation specifications for that data, write code that validates the user entries.
3. Given the Java code for an application that uses any of the language elements presented in this chapter, explain what each statement in the application does.
4. Given the output for an unhandled exception, determine the cause of the exception.

Objectives (cont.)

Knowledge

1. Explain what an access modifier is and how it affects the static methods you define.
2. Explain what the signature of a method is.
3. Explain what an exception is in Java.
4. Describe the Exception hierarchy and name two of its subclasses.
5. Explain what the stack trace is and how you can use it to determine the cause of an exception.
6. Explain how you use the try statement to catch an exception.
7. Explain what an exception handler is and when the code in an exception handler is executed.
8. Explain how you can use methods of the Scanner class to validate data.

Objectives (cont.)

9. Explain why it's usually better to validate user entries than to catch and handle exceptions caused by invalid entries.
10. Describe two types of data validation that you're likely to perform on a user entry.
11. Explain why you might want to use generic methods for data validation.

The basic syntax for coding a static method

```
public|private static returnType methodName(
        [parameterList]) {
    statements
}
```

A static method with no parameters and no return type

```
private static void printWelcomeMessage() {
    System.out.println("Hello New User");
}
```

A static method with three parameters that returns a double value

```
public static double calculateFutureValue(
        double monthlyInvestment,
        double monthlyInterestRate, int months) {
    double futureValue = 0.0;
    for (int i = 1; i <= months; i++) {
        futureValue = (futureValue + monthlyInvestment) *
                      (1 + monthlyInterestRate);
    }
    return futureValue;
}
```

The syntax for calling a static method that's in the same class

```
methodName([argumentList])
```

A call statement with no arguments

```
printWelcomeMessage();
```

A call statement that passes three arguments

```
double futureValue =
    calculateFutureValue(investment, rate, months);
```

The Future Value application with a static method

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp {

    public static void main(String[] args) {
        System.out.println(
            "Welcome to the Future Value Calculator\n");
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get the input from the user
            System.out.print(
                "Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
            double interestRate = sc.nextDouble();
            System.out.print(
                "Enter number of years:      ");
            int years = sc.nextInt();
```

The Future Value app with a static method (cont.)

```
// convert yearly values to monthly values
double monthlyInterestRate =
    interestRate/12/100;
int months = years * 12;

// call the future value method
double futureValue =
    calculateFutureValue(monthlyInvestment,
                         monthlyInterestRate,
                         months);

// format and display the result
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
System.out.println("
    Future value:               " +
    currency.format(futureValue));
System.out.println();
```

The Future Value app with a static method (cont.)

```
            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }

    // a static method that requires three arguments
    // and returns a double
    public static double calculateFutureValue(
            double monthlyInvestment,
            double monthlyInterestRate, int months) {
        double futureValue = 0.0;
        for (int i = 1; i <= months; i++) {
            futureValue =
                (futureValue + monthlyInvestment) *
                (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```

The Guess the Number application with static methods

```
import java.util.Scanner;

public class GuessNumberApp {

    private static void displayWelcome(int limit) {
        System.out.println("Guess the number!");
        System.out.println(
            "I'm thinking of a number from 1 to " + limit);
        System.out.println();
    }

    public static int getRandomInt(int limit) {
        double d = Math.random() * limit;
        int randomInt = (int) d;
        randomInt++;
        return randomInt;
    }
```

The Guess the Number application with static methods (cont.)

```
public static void main(String[] args) {
    final int LIMIT = 10;

    displayWelcome(LIMIT);
    int number = getRandomInt(LIMIT);

    Scanner sc = new Scanner(System.in);
    int count = 1;
    while (true) {
        System.out.print("Your guess: ");
        int guess = sc.nextInt();

        if (guess < 1 || guess > LIMIT) {
            System.out.println(
                "Invalid guess. Try again.");
            continue;
        }
```

The Guess the Number application with static methods (cont.)

```
            if (guess < number) {
                System.out.println("Too low.");
            } else if (guess > number) {
                System.out.println("Too high.");
            } else {
                System.out.println("You guessed it in " +
                                   count + " tries.\n");
                break;
            }
            count++;
        }
        System.out.println("Bye!");
    }
}
```

Some of the classes in the Exception hierarchy

```
Exception
    RuntimeException
        NoSuchElementException
            InputMismatchException
        IllegalArgumentException
            NumberFormatException
        ArithmeticException
        NullPointerException
```

The console with an InputMismatchException

```
Enter subtotal:   $100
Exception in thread "main"
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextDouble(Scanner.java:2456)
    at InvoiceApp.main(InvoiceApp.java:20)
```

Four methods that might throw an exception

Class	Method	Throws
`Scanner`	`nextInt()`	`InputMismatchException`
`Scanner`	`nextDouble()`	`InputMismatchException`
`Integer`	`parseInt(String)`	`NumberFormatException`
`Double`	`parseDouble(String)`	`NumberFormatException`

The syntax for a simple try/catch statement

```
try { statements }
catch (ExceptionClass exceptionName) { statements }
```

How to import the InputMismatchException class

```
import java.util.InputMismatchException;
```

A try/catch statement that catches an InputMismatchException

```
while (choice.equalsIgnoreCase("y")) {
    double subtotal = 0.0;
    try {
        System.out.print("Enter subtotal:   ");
        subtotal = sc.nextDouble();
    } catch (InputMismatchException e)  {
        System.out.println(
            "Error! Invalid number. Try again.\n");
        sc.nextLine();  // discard all data entered
                        // by the user
        continue;       // jump to the top of the loop
    }
    .
    .
}
```

Console output for the InputMismatchException

```
Enter subtotal:   $100
Error! Invalid number. Try again.

Enter subtotal:
```

The Future Value application with exception handling

```
import java.text.NumberFormat;
import java.util.Scanner;
import java.util.InputMismatchException;

public class FutureValueApp {

    public static void main(String[] args) {
        System.out.println(
            "Welcome to the Future Value Calculator\n");
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            double monthlyInvestment;
            double interestRate;
            int years;
            try {
                System.out.print("Enter monthly investment:   ");
                monthlyInvestment = sc.nextDouble();
                System.out.print( "Enter yearly interest rate: ");
                interestRate = sc.nextDouble();
```

The Future Value application with exception handling (cont.)

```
        System.out.print(
            "Enter number of years:      ");
        years = sc.nextInt();
    } catch (InputMismatchException e) {
        System.out.println(
            "Error! Invalid number. Try again.\n");
        sc.nextLine();   // discard the invalid number
        continue;        // jump to start of loop
    }

    // calculate future value
    double monthlyInterestRate = interestRate / 12 / 100;
    int months = years * 12;
    double futureValue = calculateFutureValue(
            monthlyInvestment, monthlyInterestRate, months);

    // format and display the result
    NumberFormat currency =
        NumberFormat.getCurrencyInstance();
    System.out.println("Future value:                "
            + currency.format(futureValue) + "\n");
```

The Future Value application with exception handling (cont.)

```
            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }

    private static double calculateFutureValue(
            double monthlyInvestment, double monthlyInterestRate,
            int months) {
        double futureValue = 0;
        for (int i = 1; i <= months; i++) {
            futureValue = (futureValue + monthlyInvestment) *
                          (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```

Methods of the Scanner class for validating data

- `hasNext()`
- `hasNextInt()`
- `hasNextDouble()`
- `hasNextLine()`

Code that prevents an InputMismatchException

```
while (choice.equalsIgnoreCase("y")) {
    double subtotal = 0.0;
    System.out.print("Enter subtotal:   ");
    if (sc.hasNextDouble()) {
        subtotal = sc.nextDouble();
        sc.nextLine();    // discard other entered data
    } else {
        System.out.println(
            "Error! Invalid number. Try again.\n");
        sc.nextLine();    // discard the entire line
        continue;         // jump to the top of the loop
    }
    .
    .
}
```

Console output

```
Enter subtotal:   $100
Error! Invalid number. Try again.

Enter subtotal:
```

Code that prevents a NullPointerException

```
if (customerType != null) {
    if (customerType.equals("R"))
        discountPercent = .4;
}
```

Code that gets a valid double value within a specified range

```
Scanner sc = new Scanner(System.in);
double subtotal = 0.0;
boolean isValid = false;

while (!isValid) {
    // get a valid double value
    System.out.print("Enter subtotal:   ");
    if (sc.hasNextDouble()) {
        subtotal = sc.nextDouble();
        isValid = true;
    } else {
        System.out.println(
            "Error! Invalid number. Try again.");
    }
    sc.nextLine();
            // discard any other data entered on the line
```

Code that gets a valid double value within a specified range (cont.)

```
        // check the range of the double value
        if (isValid && subtotal <= 0) {
            System.out.println(
                "Error! Number must be greater than 0.");
            isValid = false;
        } else if (isValid && subtotal >= 10000) {
            System.out.println(
                "Error! Number must be less than 10000.");
            isValid = false;
        }
    }

    System.out.println("Subtotal: " + subtotal);
```

A method that gets a valid numeric format

```
public static double getDouble(Scanner sc, String prompt)
{
    double d = 0;
    boolean isValid = false;
    while (!isValid) {
        System.out.print(prompt);
        if (sc.hasNextDouble()) {
            d = sc.nextDouble();
            isValid = true;
        } else {
            System.out.println(
                "Error! Invalid number. Try again.");
        }
        sc.nextLine();  // discard any other data
    }
    return d;
}
```

A method that checks for a valid numeric range

```
public static double getDoubleWithinRange(Scanner sc,
        String prompt, double min, double max) {
    double d = 0;
    boolean isValid = false;
    while (!isValid) {
        d = getDouble(sc, prompt);
        if (d <= min) {
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        } else if (d >= max) {
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        } else {
            isValid = true;
        }
    }
    return d;
}
```

Code that uses these methods to return two valid double values

```
Scanner sc = new Scanner(System.in);
double subtotal1 = getDouble(sc, "Enter subtotal: ");
double subtotal2 = getDoubleWithinRange(
    sc, "Enter subtotal: ", 0, 10000);
```

The console for the Future Value application with validation

```
Welcome to the Future Value Calculator

DATA ENTRY
Enter monthly investment: $100
Error! Invalid number. Try again.
Enter monthly investment: 100 dollars
Enter yearly interest rate: 120
Error! Number must be less than 30.0.
Enter yearly interest rate: 12.0
Enter number of years: one
Error! Invalid integer. Try again.
Enter number of years: 1

FORMATTED RESULTS
Monthly investment:     $100.00
Yearly interest rate:   12.0%
Number of years:        1
Future value:           $1,280.93

Continue? (y/n):
```

The code for the Future Value app with validation

```
import java.text.NumberFormat;
import java.util.Scanner;

public class FutureValueApp {

    public static void main(String[] args) {
        System.out.println(
            "Welcome to the Future Value Calculator\n");
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get the input from the user
            System.out.println("DATA ENTRY");
            double monthlyInvestment =
                getDoubleWithinRange(sc,
                    "Enter monthly investment: ", 0, 1000);
            double interestRate = getDoubleWithinRange(sc,
                    "Enter yearly interest rate: ", 0, 30);
            int years = getIntWithinRange(sc,
                    "Enter number of years: ", 0, 100);
            System.out.println();
```

The Future Value app with validation (cont.)

```
// calculate the future value
double monthlyInterestRate =
    interestRate / 12 / 100;
int months = years * 12;
double futureValue = calculateFutureValue(
    monthlyInvestment, monthlyInterestRate,
    months);

// get the currency and percent formatters
NumberFormat c =
    NumberFormat.getCurrencyInstance();
NumberFormat p =
    NumberFormat.getPercentInstance();
p.setMinimumFractionDigits(1);
```

The Future Value app with validation (cont.)

```
            // format the result as a single string
            String results
                = "Monthly investment:   "
                + c.format(monthlyInvestment) + "\n"
                + "Yearly interest rate: "
                + p.format(interestRate / 100) + "\n"
                + "Number of years:      " + years + "\n"
                + "Future value:         "
                + c.format(futureValue) + "\n";

            // print the results
            System.out.println("FORMATTED RESULTS");
            System.out.println(results);

            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            sc.nextLine();          //discard any other data
            System.out.println();
        }
    }
```

The Future Value app with validation (cont.)

```
public static double getDoubleWithinRange(Scanner sc,
        String prompt, double min, double max) {
    double d = 0;
    boolean isValid = false;
    while (!isValid) {
        d = getDouble(sc, prompt);
        if (d <= min) {
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        } else if (d >= max) {
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        } else {
            isValid = true;
        }
    }
    return d;
}
```

The Future Value app with validation (cont.)

```
public static double getDouble(Scanner sc,
        String prompt) {
    double d = 0;
    boolean isValid = false;
    while (!isValid) {
        System.out.print(prompt);
        if (sc.hasNextDouble()) {
            d = sc.nextDouble();
            isValid = true;
        } else {
            System.out.println(
                "Error! Invalid number. Try again.");
        }
        sc.nextLine();  // discard any other data
    }
    return d;
}
```

The Future Value app with validation (cont.)

```
public static int getIntWithinRange(Scanner sc,
        String prompt, int min, int max) {
    int i = 0;
    boolean isValid = false;
    while (!isValid) {
        i = getInt(sc, prompt);
        if (i <= min) {
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        } else if (i >= max) {
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        } else {
            isValid = true;
        }
    }
    return i;
}
```

The Future Value app with validation (cont.)

```
public static int getInt(Scanner sc, String prompt) {
    int i = 0;
    boolean isValid = false;
    while (!isValid) {
        System.out.print(prompt);
        if (sc.hasNextInt()) {
            i = sc.nextInt();
            isValid = true;
        } else {
            System.out.println(
                "Error! Invalid integer. Try again.");
        }
        sc.nextLine();  // discard any other data
    }
    return i;
}
```

The Future Value app with validation (cont.)

```
    public static double calculateFutureValue(
            double monthlyInvestment,
            double monthlyInterestRate, int months) {
        double futureValue = 0;
        for (int i = 1; i <= months; i++) {
            futureValue
                = (futureValue + monthlyInvestment)
                * (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```