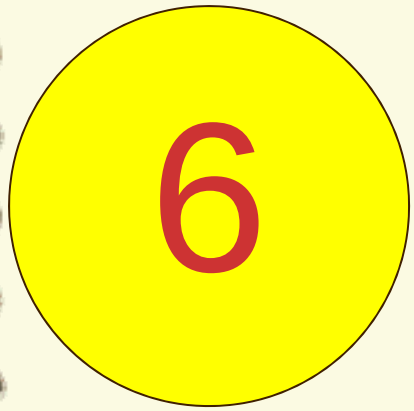
COMSC-051

Java Programming Part 1

Part-Time Instructor: Joenil Mistal

Chapter 6

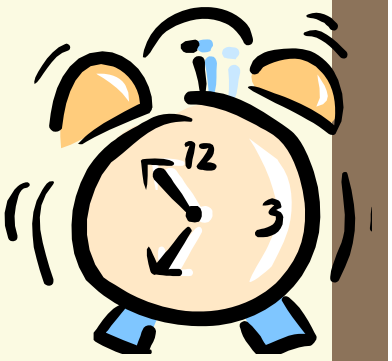


Handling Exceptions and Debugging

This chapter offers you the tools and techniques needed to handle many of the errors that might begin to pop-up as your program start to become more complex.

Chapter 6 Topics:
What you will learn in this chapter:

- What kinds of errors can occur in programming.
- How to find errors in your program
- How to handle exceptions that could crash your program
- How to test your program



Recognizing Error Types page 172

- Errors are almost unavoidable when programming.
- The programming mistakes can be referred to as bugs, errors or exceptions.
- 3 Main Categories of Errors

Syntax Error

Runtime Error

Logical Error

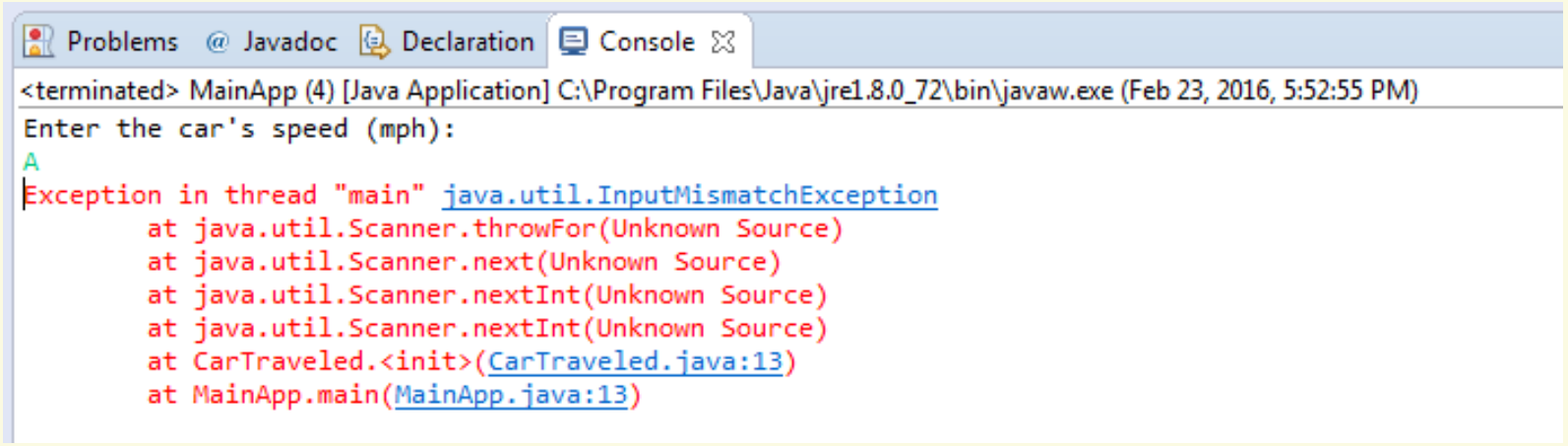
Identifying Syntax Errors page 172

- **Syntax errors** are the programming equivalent of spelling and grammar mistakes in natural languages
- **Syntax errors** include the following categories
  - Misspelled class, variable or method names
  - Misspelled keywords
  - Missing semicolons
  - Missing return type for methods
  - Out of place or mismatched parentheses & brackets
  - Undeclared or uninitialized variables
  - Incorrect format of loops, methods or other structures.

Identifying Runtime Errors

page 175

- In the console below, you'll find **red** text indicating there was an exception in the thread main.



Identifying Runtime Errors

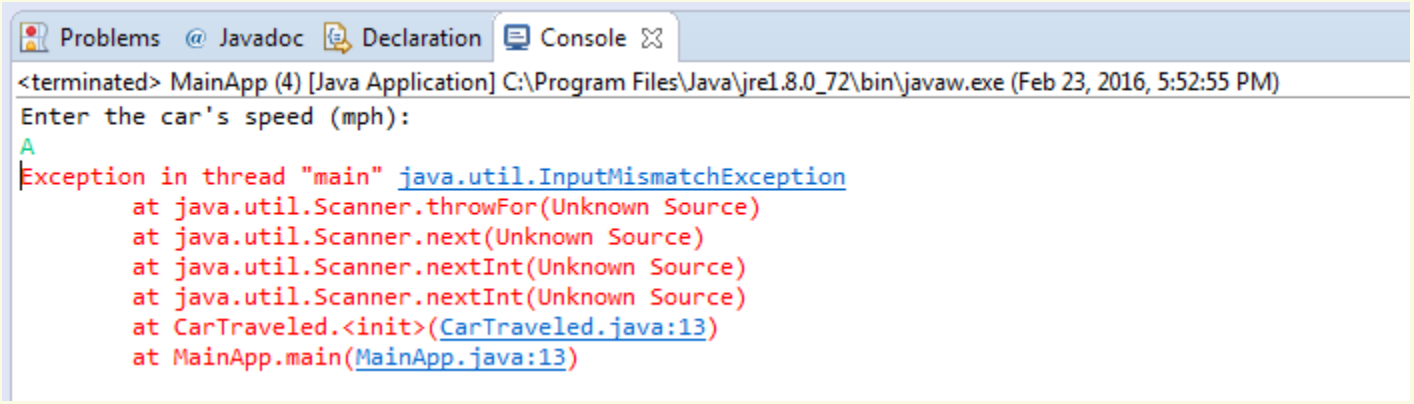
page 175

- Exception is a Java class including many types of runtime problems.
- Java distinguishes exceptions and errors:
  1. **Exceptions** can and should be managed by the programmer
  2. **Errors** are serious problems that reasonable programs are not expected to handle.

Identifying Runtime Errors

page 175

- **Division by zero** is classic runtime error example, because the syntax is correct and it is only during runtime that problem occurs.
- Another example is **input mismatch** where the program to receive a numeric value but receive a character or string.



Identifying Logical Errors

page 176

- **Logical errors** are perhaps the most difficult to spot
- The code compiles and executes without error but logically produces a wrong result.
- To detect logical errors, you will need to compare the actual results to the expected results.
- Example of Logical Error

```
33
34      // create instance method that will calculate the miles per gallon
35      public double calculateMPG (){
36          MPG = gallons / miles;
37          return MPG;
38      }
```

Problems Javadoc Declaration Console

```
<terminated> MainApp [Java Application] C:\Program Files\Java\jre1.8.0_72\bin\javaw.exe (Feb 23, 2016, 6:10:04 PM)
Enter the car model: Toyota Camary
Enter the number of miles the car can be driven on a full tank: 300
Enter gallons of gas the car can hold:  30
Miles Per Gallon = 0.10
```

Exceptions

page 180

- **Exceptions** are more specific to programming as they are events that disrupt the execution of a program.
- Exceptions can be indications that something went wrong
- They can happen automatically as a result of something Java is unable to complete or **thrown** when certain conditions are met.

Common Exceptions

page 181

- There are some exceptions that occur often
- Below are two most **common exceptions** that come up all the time.

Null Pointer Exception

Index Out of Bounds Exceptions

Common Exceptions

page 181

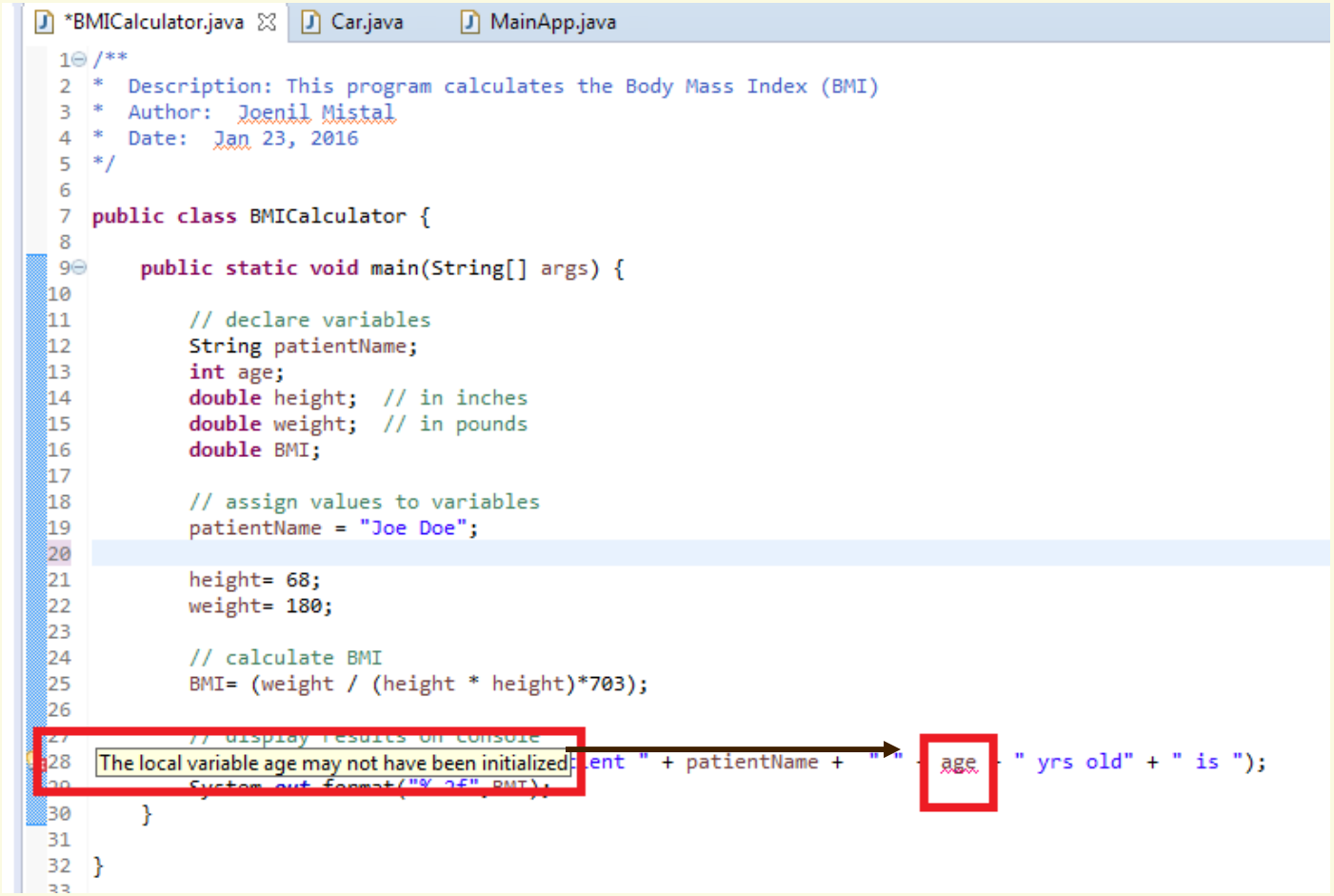
- **Null pointer exception** indicate the program is trying to access an object that does not exist yet.
- Before you can reference a data type, like int, you need to initialize it, which is setting the value.
- If you forget to do this, Eclipse will warn you.

Null Pointer Exception

Common Exceptions

page 181

- Example of a Null pointer exception



Common Exceptions

page 185

- An **Index Out of Bounds Exception** can occur when you have an indexed object such as array, and you try to access an element outside the limits of the array.
- As you've already seen, arrays are indexed starting from 0, so the last element is one less than the size of the array.

Index Out of Bounds Exceptions

Common Exceptions

page 81

- Example of Index Out of Bounds Exception

```
4  public class GrossPay {
5      // declare variables
6      double hoursWorked;
7      final static double HOURL_PAY_RATE=14;  // initialize static variable
8      final static int MAX_EMPLOYEES= 5;      // initialize static variable
9
10     // declare array to store employee's gross pay
11     double employeePayArray[] = new double [MAX_EMPLOYEES];
12
13     // create constructor(special method) to input hours earned for each employee
14     GrossPay() {
15         // declare the scanner object used to input data from console
16         Scanner inputEmpHours = new Scanner(System.in);
17
18         // used for loop to perform the iteration to input hours worked for each employee
19         // syntax of the for loop is (initialization, termination, increment)
20         for (int i = 0; i <= MAX_EMPLOYEES; i++) {
21             System.out.print("Enter hours worked for employee " + (i+1)+": " );
22             hoursWorked = inputEmpHours.nextDouble();
23
24             // calculate gross pay and store the employees pay in array
25             employeePayArray[i] = hoursWorked * HOURL_PAY_RATE;
26         }
```

Problems | Javadoc | Declaration | Console | Debug

```
<terminated> MainApp (19) [Java Application] C:\Program Files\Java\jre1.8.0_72\bin\javaw.exe (Feb 23, 2016, 7:36:09 PM)
Enter hours worked for employee 1: 10
Enter hours worked for employee 2: 20
Enter hours worked for employee 3: 30
Enter hours worked for employee 4: 40
Enter hours worked for employee 5: 50
Enter hours worked for employee 6: 10
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at GrossPay.<init>(GrossPay.java:25)
        at MainApp.main(MainApp.java:16)
```

Size of the Array is 5 but reached array index out of bounds

Catching Exceptions

page 187

- **Try-Catch Block Statement** to handle errors and exceptions
- This allows you to **try** executing a piece of code to see if an exception is thrown.
- If none is thrown, the program will proceed normally
- But if one is thrown, you can **catch** it and specifically indicate what should be done next.
- This prevents the program from crashing and at least allows you to recover some information before it terminates.

Catching Exceptions

page 187

- **Try-Catch Block Statement**

```
try {
    // execute some statements

} catch (Exception e) {
    // statements to handle the exception
} finally {
    // no mather what, do this
}
```

Catching Exceptions

page 187

- **Try-Catch Block Statement use in Exercise # 15 Membership Fee**

MainApp.java | Member.java

```
1  // import the scanner library class which allows data to be inputed from the console
2  import java.util.Scanner;
3
4  public class Member {
5
6      int typeOfMember=5, monthMembership;
7      double monthlyRate, totalFee, monthlyDiscount;
8
9      Member() {
10
11         Scanner inputData = new Scanner(System.in);
12
13             // error handling using the try-catch block.  If invalid data is inputed, execute the code in the catch container
14             try {
15                 System.out.print("Enter the Type of Membership (1- Adult, 2- Child 12 & under, 3- Student, 4-Senior Citizen): " );
16                 typeOfMember = inputData.nextInt();
17
18                 System.out.print("Enter the Number of Months of Membership: ");
19                 monthMembership = inputData.nextInt();
20
21             } catch (Exception e) {
22                 System.out.println("Invalid Membership Type. Member's rate, discount and total fee can not be calculated ");
23             }
24
25             inputData.close();  // close scanner object
26     }
```

Catching Exceptions

page 187

- **Try-Catch Block Statement use in Exercise # 16 Calories Burned**

MainApp.java | Calories.java

```
1  // import the scanner library class which allows data to be inputed from the console
2  import java.util.Scanner;
3  public class Calories {
4      // declare the arrays and initialize
5      int caloriesArray[] = new int [7];
6      String dayOfWeek[] = {"Mon","Tue","Wed","Thu","Fri","Sat","Sun"};
7
8      // declare the variables
9      int totalCal, avgCal, highestCal, lowestCal;
10
11
12     // method to get input and calculate the total weekly calories burned
13     int calcTotalCalories (){
14         // declare the scanner object used to input calories burned
15         Scanner inputCalories = new Scanner(System.in);
16
17         // error handling using the try-catch block.  If invalid data is inputed, execute the code in the catch container
18         try {
19             // used for loop to perform the iteration in inputing calories burned and accumulate the running total
20             // syntax of the for loop is (initialization, termination, increment)
21             for (int i = 0; i < caloriesArray.length; i++) {
22                 System.out.print("Enter the number of calories burned on " + dayOfWeek[i] + " :");
23                 caloriesArray[i] = inputCalories.nextInt();
24
25                 totalCal = totalCal + caloriesArray[i];   // running total of calories burned
26             }
27
28         } catch (Exception e) {
29             System.out.println("Invalid data entered.");
30         }
31
32
33         inputCalories.close();  // close scanner object
34         return totalCal;
35     }
```