Exception Handling

Christopher Simpkins
chris.simpkins@gatech.edu

Exceptions

An exception is

- an event that occurs during the execution of a program that disrupts the normal flow of instructions (Java Tutorial - Exceptions);
- a violation of the semantic constraints of a program;
- an object that is created when an exception occurs.

An exception is said to be

- *thrown* from the point where it occurred and
- *caught* at the point to which control is transferred (JLS §11).

The basic syntax is:

```
try {
    // Code that may throw an exception
} catch (Exception e) {
    // Code that is executed if an exception is
    // thrown in the try-block above
}
```

`try` Statements

In the example:

```
try {
    initFromFile(new File(employeeDataFile));
} catch (FileNotFoundException e) {
    System.out.println("Need an employee data file.");
    System.out.println(e.getMessage());
}
```

- `initFromFile()` declares that it `throws FileNotFoundException`
- If `initFromFile()` does in fact throw a `FileNotFoundException`, control is transferred to the catch block.
- The `FileNotFoundException` object that is thrown from `initFromFile()` is bound to the variable `e` in the catch block.
- The absence of the specified file is a violation of a semantic constraint of the `initFromFile` method (which it propagates from `FileReader` - more later).

Run-Time Exception Handling

Throwing an exception causes a nonlocal transfer of control from the point where the exception is thrown to the nearest dynamically enclosing catch clause, if any, of a try statement (§14.20) that can handle the exception.

> *A statement or expression is dynamically enclosed by a catch clause if it appears within the try block of the try statement of which the catch clause is a part, or if the caller of the statement or expression is dynamically enclosed by the catch clause. – JLS §11.3*

More simply, a statement is dynamically enclosed by a catch clause if it is

- contained within the corresponding try block of the catch clasuse, or
- within a method (or constructor) that is called within the corresponding try block of the catch clause.

Identifying a Dynamically Enclosing Catch Clause

```
public Company(String employeeDataFile) {
    employees = new ArrayList<Employee>(10);
    try {
        initFromFile(new File(employeeDataFile));
    } catch (FileNotFoundException e) {
        System.out.println("Can't run without an employee data file.");
        System.out.println(e.getMessage());
    }
}
private void initFromFile(File empData) throws FileNotFoundException {
    BufferedReader reader =
        new BufferedReader( new FileReader(empData) );
}
```

The highlighted statement in `initFromFile` is dynamically enclosed by the highlighted catch clause in the `Company` constructor because it is within a method that is called within the coresposnding try block of the catch clause (and the statement within the `FileReader` constructor that acutally throws the exceptoin is also dynamically enclosed by the highlighted catch clause).

Catch Block Parameters

```
public Company(String employeeDataFile) {
    employees = new ArrayList<Employee>(10);
    try {
        initFromFile(new File(employeeDataFile));
    } catch (FileNotFoundException e) {
        System.out.println("Can't run without an employee data file.");
        System.out.println(e.getMessage());
    }
}
private void initFromFile(File empData) throws FileNotFoundException {
    BufferedReader reader =
        new BufferedReader( new FileReader(empData) );
}
```

- If `new FileReader(empData)` throws a `FileNotFoundException`, it will be caught in the catch block and bound to the catch block's parameter `e`.
- The object `e` has type `FileNotFoundException` and can be used just like any other object.
- `FileNotFoundException` is a standard library exception, so

Checked and Unchecked Exceptions

Checked exceptions are subclasses of `Throwable` that are not subclasses of `RuntimeException` or `Error`. The compiler requires that checked exceptions declared in the throws clauses of methods are handled by:

- a dynamically enclosing catch clause, or
- a `throws` declaration on the enclosing method or constructor.

This rule is sometimes called “catch or specify” or “catch or declare.”

Unchecked exceptions (subclasses of `RuntimeException` or `Error`) are not subject to the catch or declare rule.

Catch or Declare

For example, here are the two ways to deal with the `FileNotFoundException` thrown by `initFromFile`.

Catch:

```
public Company(String employeeDataFile) {
  // ...
  try {
    initFromFile(new File(employeeDataFile));
  } catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
  }
}
```

Declare:

```
public Company(String employeeDataFile) throws FileNotFoundException {
  // ...
  initFromFile(new File(employeeDataFile));
}
```

Taking Advantage of Unchecked Exceptions

We've been using the Scanner class without having to handle exceptions because its methods throw unchecked exceptions. But we can make use of these exceptions to make our code more robust.

```
Scanner kbd = new Scanner(System.in);
int number = 0;
boolean isValidInput = false;
while (!isValidInput) {
    try {
        System.out.print("Enter an integer: ");
        number = kbd.nextInt();
        // If nextInt() throws an exception, we won't get here
        isValidInput = true;
    } catch (InputMismatchException e) {
        // This nextLine() consumes the token that
        // nextInt() couldn't translate to an int.
        String input = kbd.nextLine();
        System.out.println(input + " is not an integer.");
        System.out.println("Try again.");
    }
}
```

Multiple Catch Clauses

Two important points in writing multiple catch clauses for a try statement:

- The exception type in a catch clause matches subclasses.
- The first catch clause that matches an exception is the (only) one that executes.

This means that you should order your catch clauses from most specific (most derived, lowest in exception class hierarchy) to least specific (highest in exception class hierarchy).

Common Mistakes in Multiple Catch Clauses

What's wrong with this code?

```
try {
  initFromFile(new File(employeeDataFile));
} catch (Exception e) {
  System.out.println("Something bad happened: " + e.getMessage());
} catch (IOException e) {
  System.out.println(e.getMessage());
  e.printStackTrace(System.out);
  System.exit(0);
} catch (FileNotFoundException e) {
  System.out.println("Can't run without an employee data file.");
  System.out.println(e.getMessage());
  System.exit(0);
} catch (ParseException e) {
  String msg = "Malformed data caused exception: " + e.getMessage();
  System.out.println(msg);
  System.out.println("Full stack trace:");
  e.printStackTrace(System.out);
  System.exit(0);
}
```

Ivestigative Exercise

Let's conclude by asking ourselves a couple of questions.

- Can a method declare an unchecked exception in its `throws` clause?
- If so, does code that calls the method then have to deal with the unchecked exception as if it were checked, that is, does the catch or declare rule apply?

Let's find out!