Exceptions

Handling exception in Java programs

Checked Exceptions

- Exceptions fall into three categories
- Internal errors are reported by descendants of the type Error.
  - Example: OutOfMemoryError
- Descendants of RuntimeException,
  - Example: IndexOutOfBoundsException or IllegalArgumentException
  - They indicate errors in your code.
  - They are called unchecked exceptions.
- All other exceptions are checked exceptions.
  - Indicate that something has gone wrong for some external reason beyond your control
  - Example: IOException

Checked Exceptions

- Checked exceptions are due to external circumstances that the programmer cannot prevent.
  - The compiler checks that your program handles these exceptions.
- Handling the unchecked exceptions are the responsibility of the programmer
  - The compiler does not check whether you handle an unchecked exception.

Checked Exceptions — throws

- You can handle the checked exception in the same method that throws it

```
try
{
        File inFile = new File(filename);
        // Throws FileNotFoundException
        Scanner in = new Scanner(inFile);
        //. . .
}
catch (FileNotFoundException exception) // Exception caught here
{
        //. . .
}
```

Checked Exceptions — throws

- Often the current method cannot handle the exception. In such cases,
  - Tell the compiler you are aware of the exception
  - You want the method to terminate if the exception occurs
  - Add a throws clause to the method header

```
public void readData(String filename) throws FileNotFoundException
{
    File inFile = new File(filename);
    Scanner in = new Scanner(inFile);
    //. . .
}
```

Checked Exceptions — throws

- The throws clause signals to the caller of your method that it may encounter a FileNotFoundException.
- The caller must decide
  - To handle the exception
  - Or declare the exception may be thrown

- Throw early, catch late
  - Throw an exception as soon as a problem is detected.
  - Catch it only when the problem can be handled

Closing Resources

- Resources that must be closed require careful handling, such as PrintWriter
- Use the try-with-resources statement:
  - If no exception occurs, out.close() is called after writeData() returns
  - If an exception occurs, out.close() is called before exception is passed to its handler

```
try (PrintWriter out = new PrintWriter(filename)) {
        writeData(out);
    } // out.close() is always called
```

Designing Your Own Exception Types

- You can design your own exception types — subclasses of
- Exception or RuntimeException.
- Throw an InsufficientFundsException when the amount to withdraw an amount from a bank account exceeds the current balance.

- How do we create InsufficientFundsException as an unchecked exception class?
  - We can extend the IllegalArgumentException class

```
if (amount > balance){
    throw new
    InsufficientFundsException("withdrawal of " +
                                amount + " exceeds balance of " + balance);
}
```

Designing Your Own Exception Types

- Supply two constructors for the class
  - A constructor with no arguments
  - A constructor that accepts a message string describing reason for exception
- When the exception is caught, its message string can be retrieved by using the getMessage method

```
public class InsufficientFundsException extends IllegalArgumentException
{
    public InsufficientFundsException() {}

    public InsufficientFundsException(String message)
    {
        super(message);
    }
}
```

Self Check 11.16

- Suppose balance is 100 and amount is 300. What is the value of balance after these statements?

```
if (amount > balance)
{
 throw new IllegalArgumentException("Amount exceeds
 balance");
}
balance = balance - amount;
```

Thank you

Please let me know if you have any questions.


Questions?