Exceptions, Assertions and Logging

Yangtao Ge

August 23, 2019

This section will talk about three topics

- **Exception handling**: use some cases to avoid accidental errors
- **Assertions**: run several checks to make sure you program does the right thing
- **Logging**: record problems into files

1 Dealing with Errors

Basic Requirement:

- Return to a safe state and enable the user to execute other commands
- Allow user to save all work and terminate the program gracefully

Possible Errors:

- User input errors: *syntatically* wrong
- Device errors: Hardware may not be able to do what you want (Power off?)
- Physical limitations: Disks can be filled up

- Code errors: using something in a wrong way for existing codes

In Java, we use 'throw' to provide an object *which encapsulates the error information*

1.1 The Classification of Exception

Here is the hierarchy of Exception in Java:

Class **Throwable**:

- Error: ...
- Exception:
  - IOException:
    - read past the end of a file
    - open a file that doesn't existing
    - find a *Class* Object for a string that does not denote an existing class
  - Runtime Exception: RuntimeException means it was your fault
    - A bad cast
    - An out of bounds array access
    - A null pointer access

Here is the classification of Exception:

- **unchecked** Exception:
  - class **Error**
  - class **RuntimeException**
- **checked** Exception: *Others*

1.2 Declaring Checked Exception

Aim of throwing exceptions:

- tell the Java Compiler what values it can return
- tell the Java Compiler what can go wrong

We can throw an exception from a method or a constructor(Class).

For a constructor, when we intialize an object, either it will produce an object correctly, or it will thrown an Exception object. This is the same as methods.

For a method, we **do not** need to throw every possible exceptions. Here are 4 situations where exception will be thrown:

- Call a method that throws a **checked** exception
- Detect an **error** or a **checked** exception with the '*throw*' statement
- Make a programming error which rise to an **unchecked** exception
- An internal error occurs in the virtual machine or runtime library.

In summary:

- a method must declare all the checked exception that it might throw.
- unchecked exceptions are either beyond your control or result from conditions that we should not allow in the first place

1.3 How to Throw an Exception

An example for how to throw an exception:

```
String readData(Scanner in) throws EOFException{
    ...
    while(...){
        if(!in.hasNext()){
            if(n<len) throw new EOFException();
        }
    }
    return s;
}
```

and we could define the exception message if we wish

```
String gripe = ``content-length: "+ len +``, Received: " + n;
throw new EOFException(gripe);
```

1.4 Creating Exception Classes

we can create our own exception classes:

```
    class FileFormatException extends IOException{
        public FileFormatException(){}
        public FileFormatException(){
            super(gripe);
        }
    }
```

2 Catching Exceptions

Catching needs more <u>plans</u> than throwing

2.1 Catching an Exception