OOP using Java

Trainer: Mr. Rohan Paramane

Agenda

- Exception Handling
- Package using command line

Error

- java.lang.Error is a sub class of Throwable class.
- It gets generated due to environmental condition/Runtime environment (For Example, problem in RAM/JVM, Crashing HDD etc.).
- We can not recover from error hence we should not try to catch error.
- But can write try-catch block to handle error.
- **Example:**
  - VirtualMachineError
  - OutOfMemoryError
  - InternalError
  - StackOverflowError

Exception

- java.lang.Exception is a sub class of Throwable class.
- Exception gets generated due to application.
- We can recover from exception hence it is recommended to write try-catch block to handle exception in Java.
- **Example:**
  - NumberFormatException
  - NullPointerException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException
  - IllegalArgumentException
  - ClassCastException

Types Of Exception

- **Unchecked Exception :**
  - java.lang.RuntimeException and all its sub classes are considered as unchecked exception.
  - It is not mandatory to handle unchecked exception.
  - **Example:**
    - NullPointerException
    - ClassCastException
    - ArrayIndexOutOfBoundsException
  - During the execution of arithmetic operation, if any exceptional situation occurs then JVM throws ArithmeticException.

- **Checked Exception :**
  - java.lang.Exception and all its sub classes except java.lang.RuntimeException are considered as checked exception.
  - It is mandatory to handle checked exception.
  - Example:
  - java.lang.CloneNotSupportedException
  - java.lang.InterruptedException

try & catch

try :

- It is a keyword in Java.
- If we want to keep watch on statements for the exception then put all such statements inside try block/handler.
- try block must have at least one:
- catch block or
- finally block or
- Resource
- We can not define try block after catch or finally block.

catch :

- It is a keyword in Java.
- If we want to handle exception then we should use catch block/handler
- Only Throwable class or one of its subclasses can be the argument type in a catch clause.
- Catch block can handle exception thrown from try block only.
- For single try block we can define multiple catch block.
- Multi-catch block allows us to handle multiple specific exception inside single catch block.

Multiple and Generic Catch Block

- In case of hierarchy, It is necessary to handle all sub type of exception first.
- A catch block, which can handle all type of exception is called generic catch block.
- Exception class reference variable can contain reference of instance of any checked as well as unchecked exception. Hence to write generic catch block, we should use java.lang.Exception class.

```
try {
    //TODO
}catch (ArithmeticException e) {
    e.printStackTrace();
}catch (RuntimeException e) {
    e.printStackTrace();
}catch (Exception e) {
    e.printStackTrace();
}
```

```
try{


}catch( Exception ex ){ //Generic catch block
    ex.printStackTrace( );
}
```

throw & throws

- **throw :**
  - It is a keyword in Java.
  - If we want to generate new exception then we should use throw keyword.
  - Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.
  - throw statement is a jump statement.

- **throws :**
  - It is a keyword in Java.
  - If we want to redirect/delegate exception from one method to another then we should use throws clause.
  - Consider declaration of following methods:
    - public static int parseInt(String s) throws NumberFormatException
    - public static void sleep(long millis) throws InterruptedException

finally & try with resource

- **finally :**
  - It is a keyword in Java.
  - If we want to release local resources then we should use finally block.
  - We can not define finally block before try and catch block.
  - Try block may have only one finally block.
  - JVM always execute finally block.
  - If we call System.exit(0) inside try block and catch block then JVM do not execute finally block.

- **try-with-resources :**
  - The try-with-resources statement is a try statement that declares one or more resources.
  - The try-with-resources statement ensures that each resource is closed at the end of the statement.
  - Any object that implements java.lang.AutoCloseable, which includes all objects that implement java.io.Closeable, can be used as a resource

Custom Exception

- JVM can not understand, exceptional situations/conditions of business logic.
- If we want to handle such exceptional conditions then we should use custom exceptions.

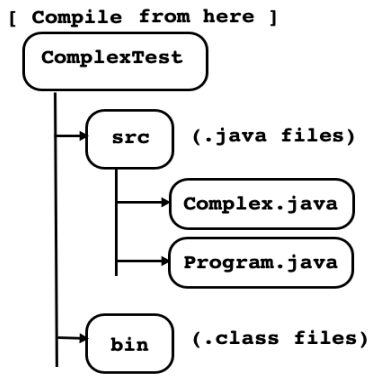
Custom unchecked exception

```
class StackOverflowException extends RuntimeException{
    //TODO
}
```

Custom checked exception

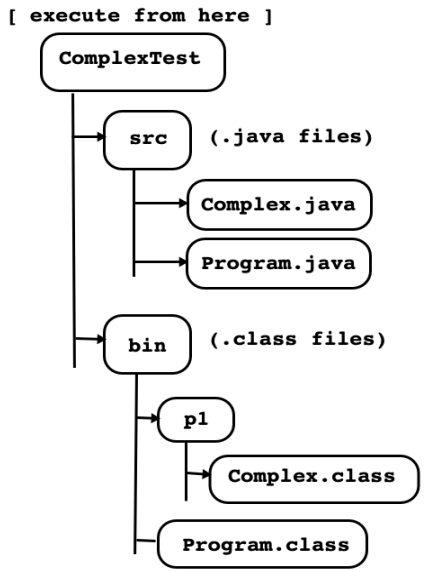
```
class StackOverflowException extends Exception{
    //TODO
}
```

Package Example

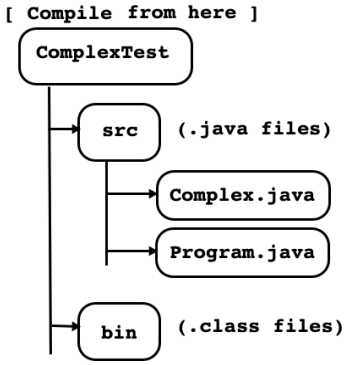


[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

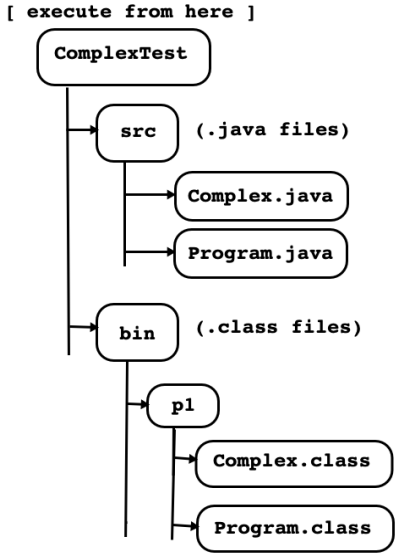


[After Compilation]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”



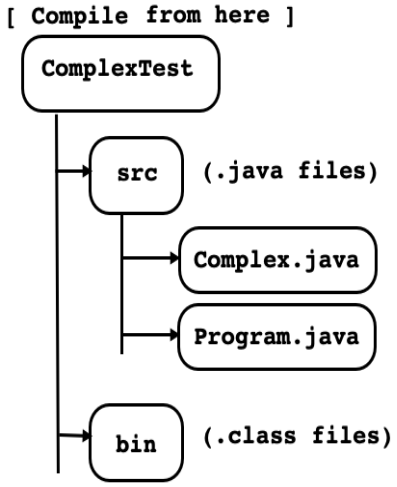
[After Compilation]

package p1
- Complex class

default package
- Program class

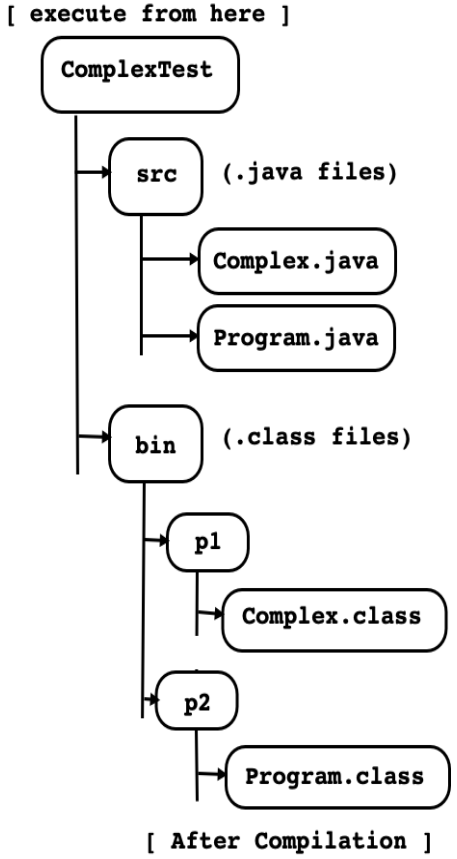
package p1
- Complex class
- Program class

Package Example



[Before Compilation]

1. Set path(if not set)
2. Compile "Complex.java"
3. set classpath
4. Compile "Program.java"
5. execute "Program.class"



[After Compilation]

package p1
- Complex class

Package p2
- Program class

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com