Error - exception handling

Object Oriented Programming with Java

Chapter 5

FPTU Da Nang – IT Department

Objectives - Content

- Error & Exception Concepts
- Try – catch – finally structure
- throw & throws clause
- Exception class definition
- Exceptions and overriding

Error handling in Java

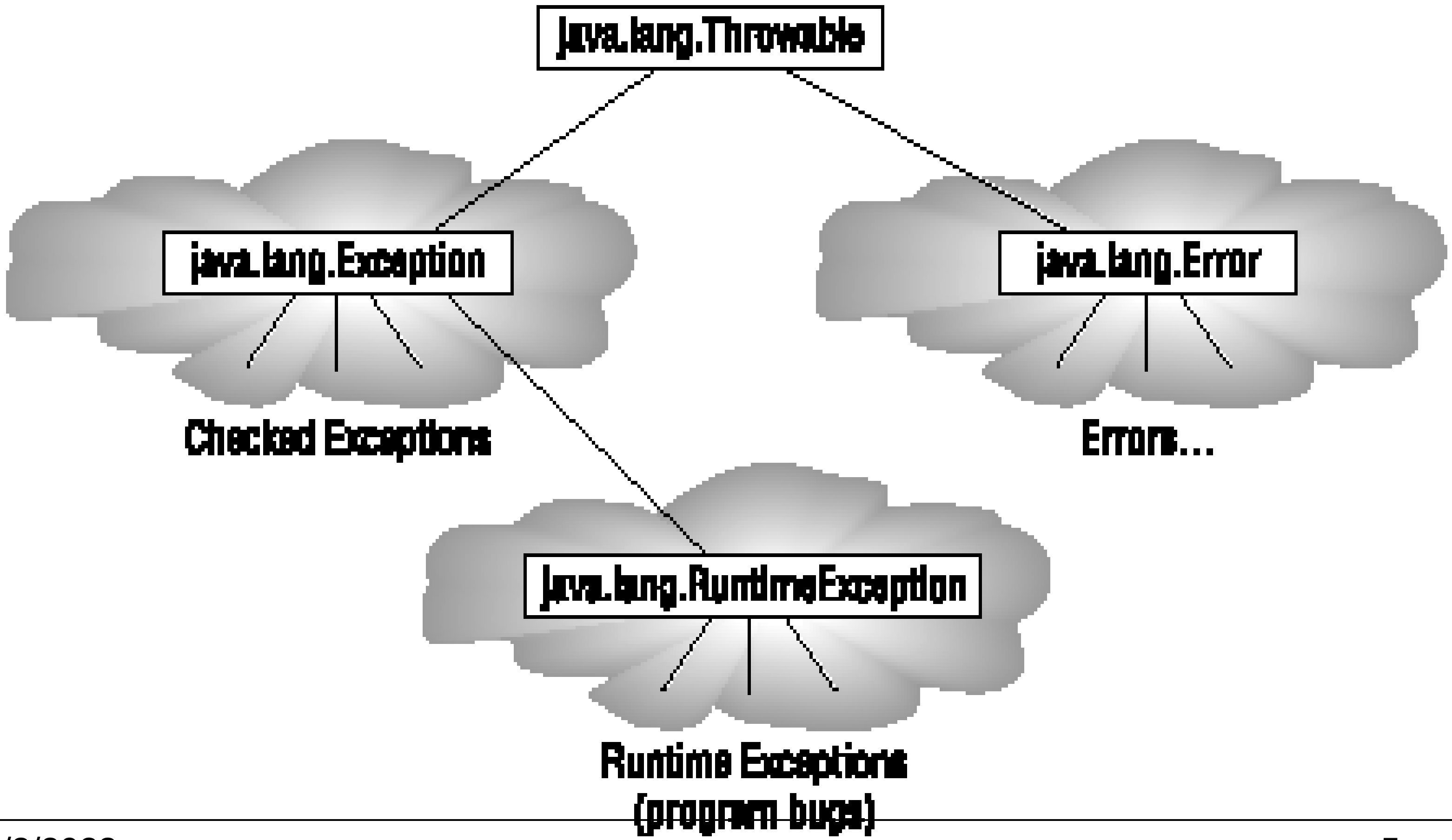
- The usual way of handling errors of programming language is that when an error is detected, it will report an error to the user or return an abnormal value.
- Java was the first programming language that handling error focus on distributed application on network environments.
- Exception - errors are propagated to the source of the error by tracing and throw - returning the error to the original place.

Error – Exception concept

- Error classification
  - Error are unusual, unpredictable, unrecoverable code incident:
    - out of memory, system error, network environment
  - Exception: possible situations that affect the normal logic of the program
    - logic error, programming error
- Error & Exception types are all represented by Error & Exception classes that inherit from **java.lang.Throwable** class
- Exception contains information about the situation, including the error type, the state of the program when the error occurs.
- The JVM system is responsible for finding the program code for exception handling

Error – Exception hierarchy

- Throwable: throw away



Fpt University

Exception

- Exception classification
  - **Runtime exceptions**:
    - Errors occur at runtime, often due to poor code writing
      + Access array out of bound
    - Predictable avoidance when writing source code
    - RuntimeException & its subclasses
    - no processing code is required
  - **Checked exceptions**
    - error has been checked, forecast
    - Exception and subclasses - except RuntimeException
    - must have handling code

Handling Exceptions

- The Java compiler requires all checked Exceptions to be handled
- Two Exception handling options:
  - catch errors and handle using blocks try - catch - finally.
  - throws error using throw and declare throws
- When an error occurs, the JVM will raise an exception, the thread executing the method will terminate the execution of the next instructions after the error situation, and this exception will be thrown towards the object calling the method.
  - if there is no handling code, control will return back to main() function.

Two error handling mechanisms

- Catch error
  - Use the **try.. catch.. finally** . construct
  - Handle errors so that the program does not terminate abnormally.
  - Allow re-try when there is an error/exception
- Throw back an error to the original place
  - Declare the method as **throws Exception** .
  - Throw an error using the **throw new Exception()**
  - Only throws an error when it's out of control

Business Exception

- In Java, creating an Exception object on error and passing it to the JVM at run time is called throwing an exception.
- Errors related to application logic should be represented by defining subclasses of Exception or RuntimeException
  - Helps distinguish business errors and system errors
  - public class BankingException extends Exception { public BankingException(String msg) {super(msg); } }

throw and throws

- Exceptions are actually thrown from method when it is necessary to notify the calling object of error situation
- Use the throw clause to throw an Exception
  - if (balance<amount) throw new BankingException(“Out of money”);
- The method containing the **throw** statement must declare the possibility of an error with the **throws** clause
  - No need to declare throws exception if the exception is a subclass of RuntimeException
  - public double withdraw(double amount) **throws** BankingException{
    if (balance<amount) **throw** new BankingException(“No money”);
    else balance-=amount;
    return balance;
    }

Use try.. catch.. finally

```
try{ doSomething; }
catch(Exception e){ doXulyLoi; }
finally { alwayDoThis; }
```

- **Try** block:
  - statements that can generate an exception.
  - will terminate the execution of the remaining instructions to jump to the catch block when an error occurs.
- **Catch** block:
  - error that handling error.
  - Error information stored in Exception e
  - If there are more than one type of Exception then multiple catch blocks can be used.
  - Cannot catch if there is no possibility of error in the try . block
- **Finally** block: (optional)
  - statements will be executed, regardless of errors or not

Use try.. catch..

- Example:

```
int x = (int)(Math.random() * 5);
int y = (int)(Math.random() * 10);
int [] z = new int[5];
try {
 System.out.println(“y/x gives “ + (y/x));
 System.out.println(“y is “+ y + “ z[y] is” + z[y]);
 }
 catch (ArithmeticException e) {
 System.out.println(“Arithmetic problem” + e);
 }
 catch (ArrayIndexOutOfBoundsException e) {
 System.out.println(“Subscript problem” + e);
 }
```

- **Try must be accompanied by catch or finally**

6. Consider the following class hierarchy and code fragment:

```
                       java.lang.Exception
                              \
                       java.io.IOException
                          /              \
java.io.StreamCorruptedException      java.net.MalformedURLException
```

```
try {
   // assume s is previously defined
   URL u = new URL(s);
   // in is an ObjectInputStream
   Object o = in.readObject();
   System.out.println("Success");
}
catch (MalformedURLException e) {
   System.out.println("Bad URL");
}
catch (StreamCorruptedException e) {
   System.out.println("Bad file contents");
}
catch (Exception e) {
   System.out.println("General exception");
}
finally {
   System.out.println("doing finally part");
```

Exceptions and Overriding

- The override method is not allowed to throw more checked exceptions than the overriden method in the base class
  - For that reason, some base classes need to declare throws an Exception even though there isn't an Exception
- The override method can throw Exceptions so that these exceptions are subclasses of the exception thrown in the overriden method.

Exception & Overriding

- **Review question**
  The risky() method might throw a java.io.IOException, java.lang.RuntimeException, java.net.MalformedURLException (which is a subclass of java.io.IOException). Which of the following classes and sets of classes are legal?
- **A. public class SomeClass {**
- **public void aMethod() {**
- **risky();**
- **}**
- **}**
- **B. public class SomeClass {**
- **public void aMethod() throws IOException {**
- **risky();**
- **}**
- **}**

Exception & Overriding

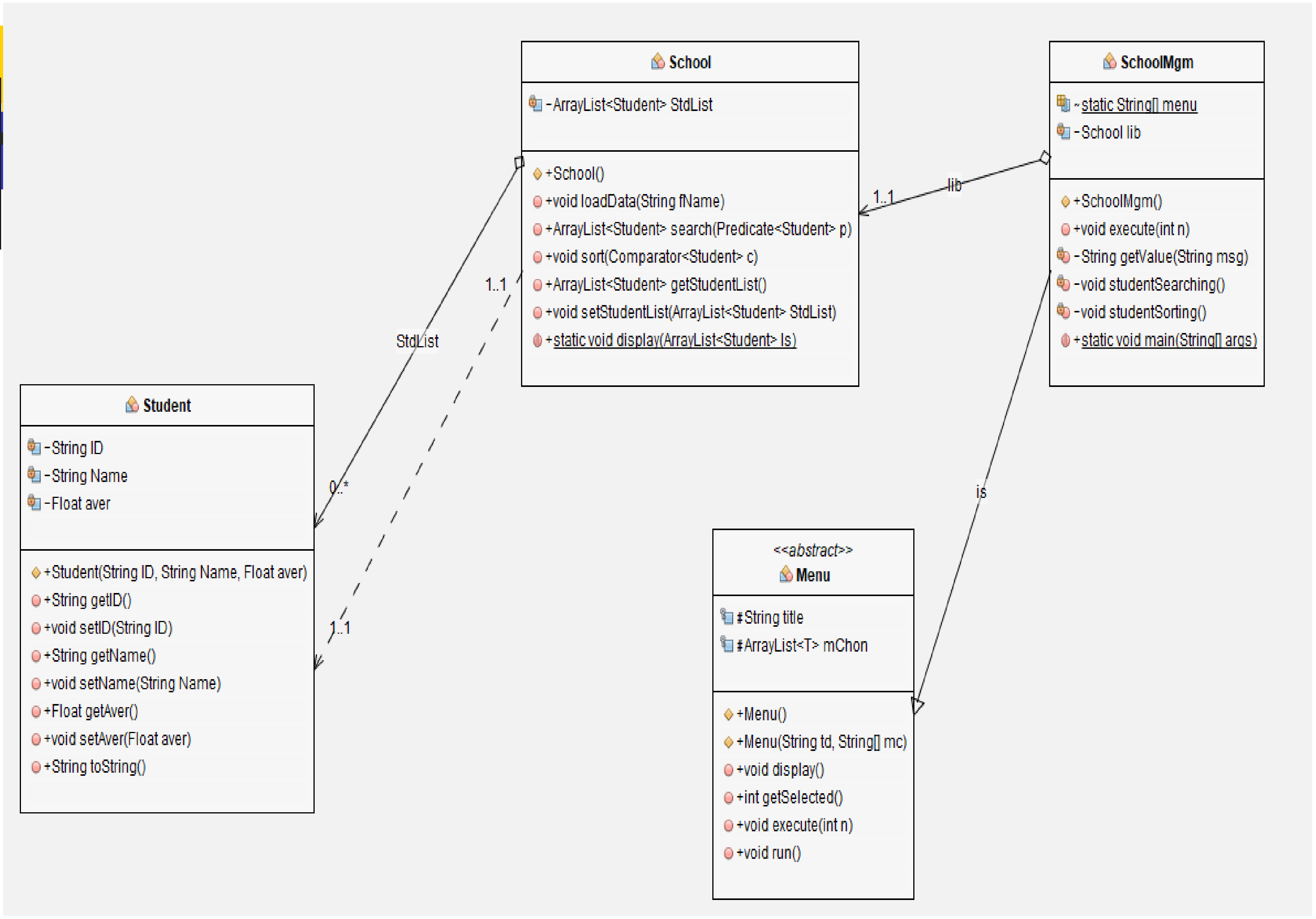
```
C. public class SomeClass {
 public void aMethod() throws RuntimeException {
 risky();
 } }
D. public class SomeClass {
 public void aMethod() {
 try {
 risky();
 }
 catch (IOException e) {
 e.printStackTrace();
 }
 }
 }
```

Exception & Overriding

- E. public class SomeClass {
- public void aMethod() throws MalformedURLException {
- try {
- risky();
- }
- catch (IOException e) { }
- }
- };
- public class AnotherClass extends SomeClass {
- public void aMethod() throws java.io.IOException {
- super.aMethod();
- }
- }

Constructive questions

- How is error handling in Java different from other languages?
- When errors should be catched, should be thrown?
- How to prevent the program from crashing because the user accidentally entered the wrong data?
- Write a function that allows input of an integer from the keyboard, valid from m to n. Any invalid input is allowed to re-enter until ok
- A function declaration throws Exception, even though the function body is not capable of generating an error. What is this for?
- Define a class of business exception of the library management application. System logging when errors occur.


School
-ArrayList<Student> StdList
+School()
+void loadData(String fName)
+ArrayList<Student> search(Predicate<Student> p)
+void sort(Comparator<Student> c)
+ArrayList<Student> getStudentList()
+void setStudentList(ArrayList<Student> StdList)
+static void display(ArrayList<Student> ls)
SchoolMgm
~static String[] menu
-School lib
+SchoolMgm()
+void execute(int n)
-String getValue(String msg)
-void studentSearching()
-void studentSorting()
+static void main(String[] args)
lib
1..1
StdList
1..1
0..*
1..1
is
Student
-String ID
-String Name
-Float aver
+Student(String ID, String Name, Float aver)
+String getID()
+void setID(String ID)
+String getName()
+void setName(String Name)
+Float getAver()
+void setAver(Float aver)
+String toString()
<<abstract>>
Menu
#String title
#ArrayList<T> mChon
+Menu()
+Menu(String td, String[] mc)
+void display()
+int getSelected()
+void execute(int n)
+void run()