EECS2030 Advanced Object-Oriented Programming (Fall 2021)

Q&A - Lecture 2a

Wednesday, September 29

Announcement

- Lab1 (due: Oct. 1) → 2pm Friday
- Written Test (due: Sep. 30 - Oct. 1) → 2pm Friday
- Lecture W4 (released: Sep. 27)
- Lab2 (to be released: Oct. 1)

In the class A, in NegValException exception,
we passed a String for initialization and when it was the abnormal case,
the part from the catch method got printed into the console, specifically,
what is the point of having an argument passed over the exception?

Version 3

```
class A
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error: "); }
    else { /* Do something. */ }
  } }
```

"Neg Val: " + i

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
class Tester {
  public static void main(String[] args) throws NegValException {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i);
  } }
```

```
Enter an integer i:
-20
Exception in thread "main" version_3.NegValException: Neg Val: -20
        at version_3.A.ma(A.java:11)
        at version_3.B.mb(B.java:12)
        at version_3.Tester.main(Tester.java:19)
```

For version 2, since we are catching the exception in Tester.main,
why did we had class B, couldn't we specify the exception directly from class A to Tester.main without having class B?

↳ In general, the size of call stack would not be just 2.

In a general case, if an exception is thrown at the top of our method stacks, shouldn't the exception be specified to the caller coming after it?

↳ it's possible for a callee's exception to be handled by where exception was thrown

Why should we just have another caller after the place where the exception was originated, throw it to its caller (2nd method call in the stack from the top) and then throw it again to its caller (3rd method call in the stack from the top)?

↳ it depends on the app. logic.

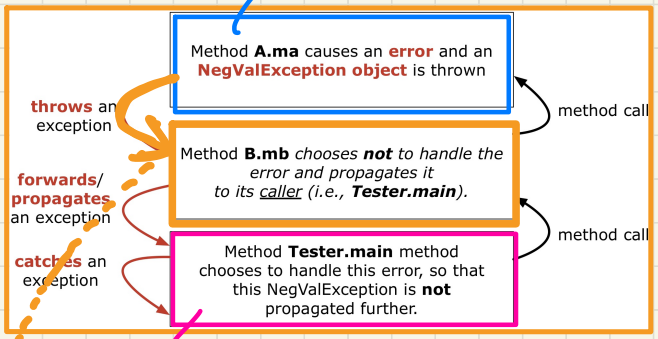
```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    try { ob.mb(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

its caller immediately but this does not happen often.

to catch or specify.



where exception was caught

```java
public class InvalidTransactionException extends Exception {
  public InvalidTransactionException(String s) {
    super(s);
  }
}
```

```java
class Account {
  int id; double balance;
  Account() { /* balance defaults to 0 */ }
  void withdraw(double a) throws InvalidTransactionException {
    if (a < 0 || balance - a < 0) {
      throw new InvalidTransactionException("Invalid withdraw."); }
    else { balance -= a; }
  }
}
```

```java
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdraw(int id, double a)
      throws InvalidTransactionException {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id) {
        accounts[i].withdraw(a);
      }
    } /* exception */ }
```

```java
class BankApplication {
  pubic static void main(String[] args) {
    Bank b = new Bank();
    Account acc1 = new Account(23);
    b.addAccount(acc1);
    Scanner input = new Scanner(System.in);
    double a = input.nextDouble();
    try {
      b.withdraw(23, a);
      System.out.println(acc1.balance); }
    catch (InvalidTransactionException e) {
      System.out.println(e); } } }
```

Account.withdraw

Bank.withdraw

BankApp.main

In a general case, if an exception is thrown at the top of our method stacks, shouldn't the exception be specified to the caller coming after it?

↳ not necessarily.

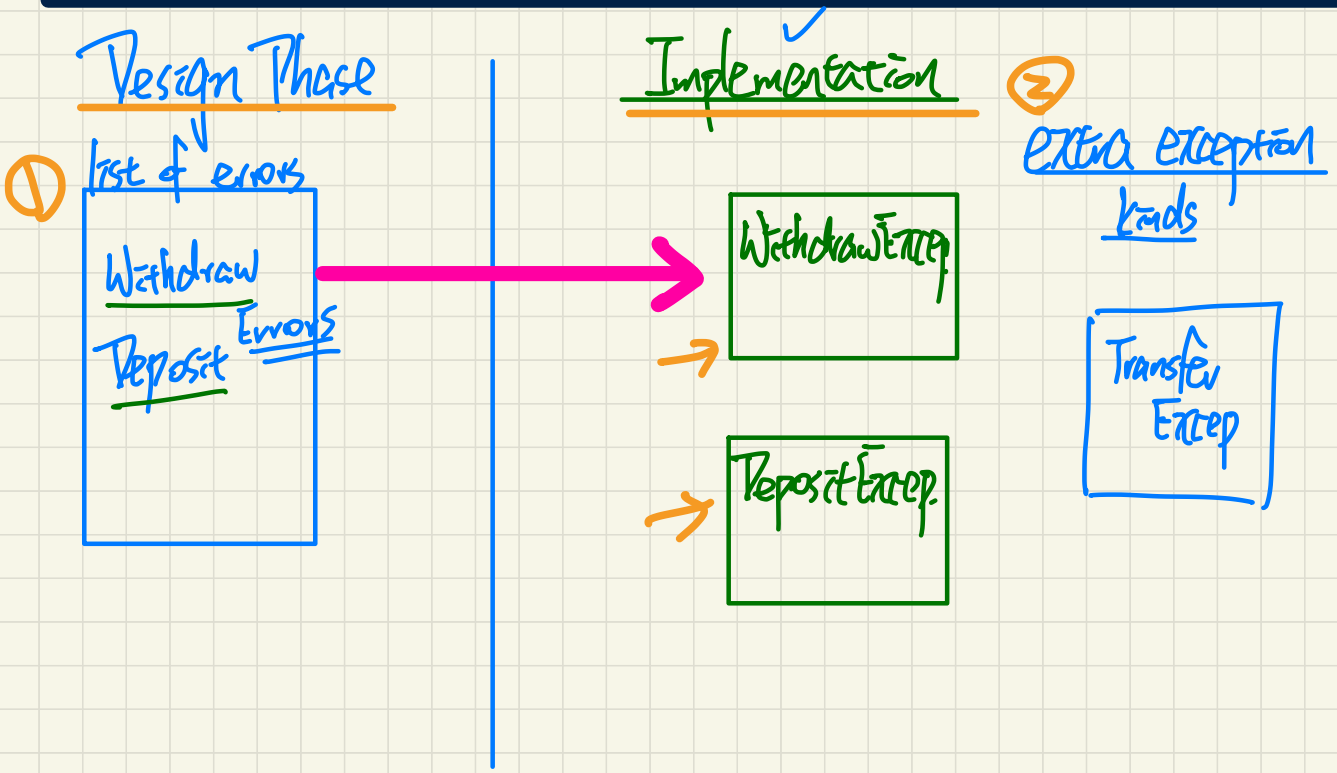
Why should we just have another caller after the place where the exception was originated, throw it to its caller (2nd method call in the stack from the top) and then throw it again to its caller (3rd method call in the stack from the top)?

Account

Origin of the invalid input

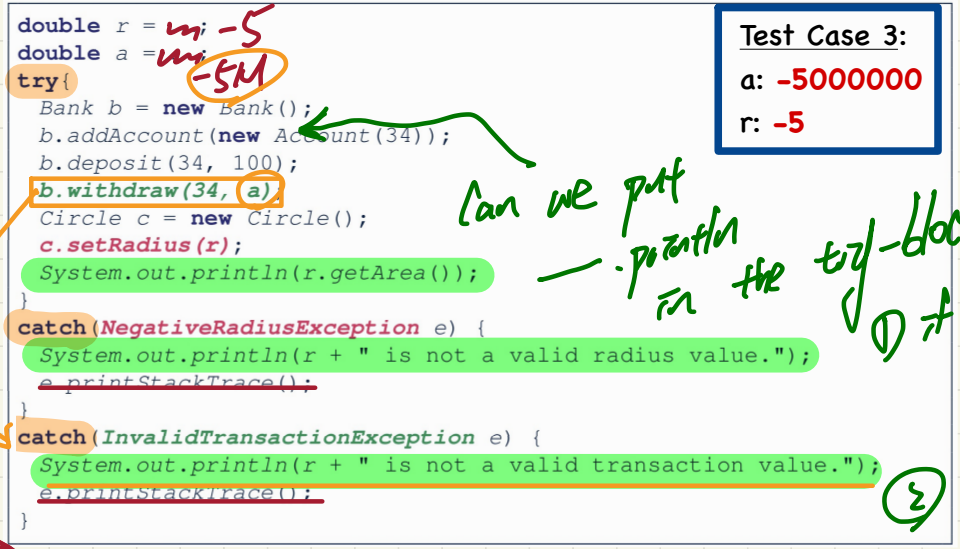
For this app, we only handle the exception in the Console app class; this is where the amount is read.

Should we do exception handling as we develop our program, or we should finish the program and then do it? Both.



As shown in this video, what if we call two or more methods inside a try block and all of their callees throw exceptions?

How can we notice that all of them threw an error; because once one of them caught an exception the rest of the try block would be bypassed?



Why do we only have 1 try block?
Can we do 2 try blocks then if the second is invalid,
the user only has to re-enter the second rather than restarting

```
double r = ...;
double a = ...;
try{
  Bank b = new Bank();
  b.addAccount(new Account(34));
  b.deposit(34, 100);
  b.withdraw(34, a);
  Circle c = new Circle();
  c.setRadius(r);
  System.out.println(r.getArea());
}
catch(NegativeRadiusException e) {
  System.out.println(r + " is not a valid radius value.");
  e.printStackTrace();
}
catch(InvalidTransactionException e) {
  System.out.println(r + " is not a valid transaction value.");
  e.printStackTrace();
}
```

Test Case 1:
a: -5000000
r: 23

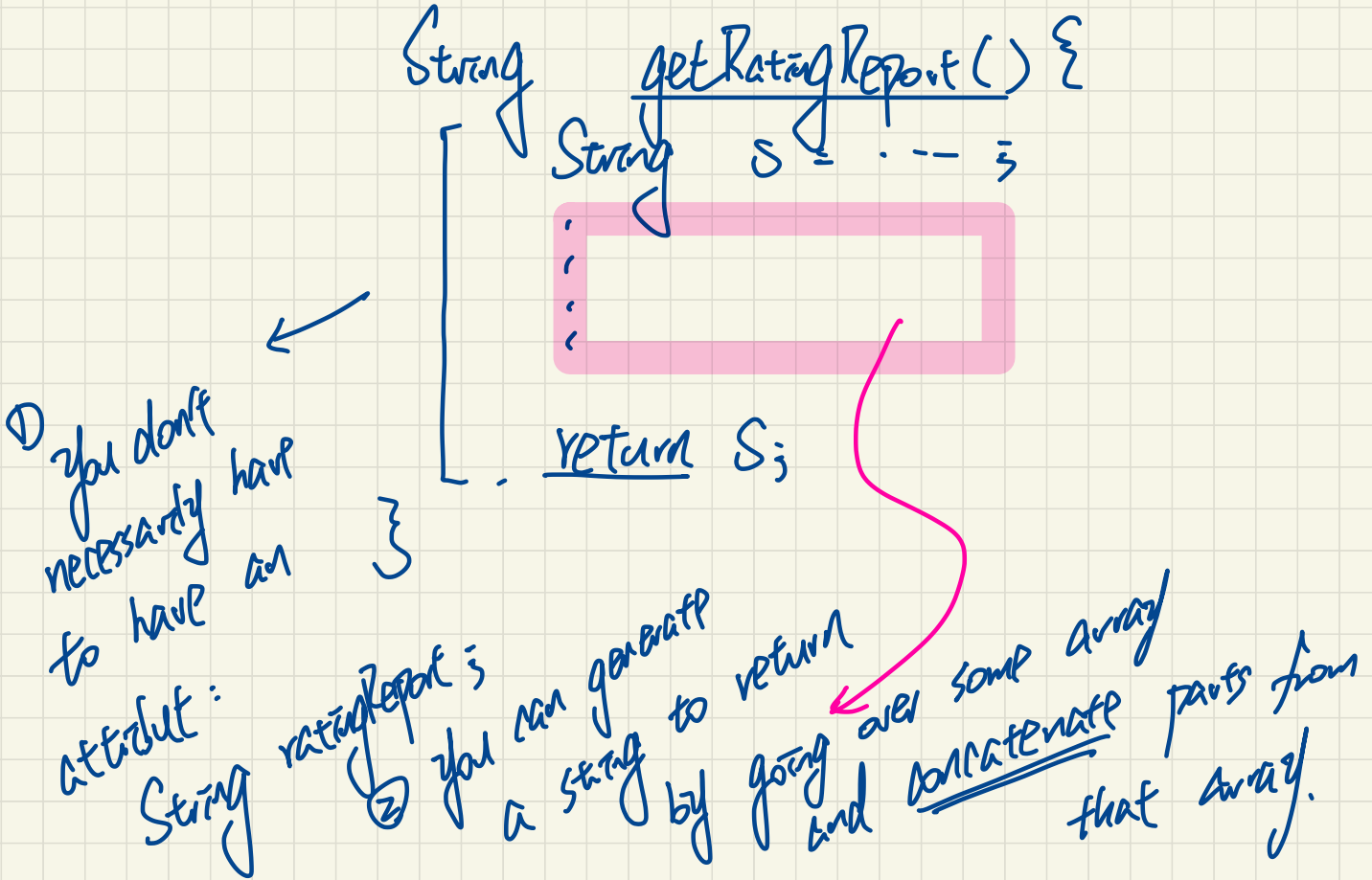
Test Case 2:
a: 100
r: -5

Version 7: multiple try-catch

```
double r = ...;
double a = ...;
try {
    Bank b = ...;
    b.addAccount(...);
    b.deposit(...);
    b.withdraw(34, a);
}
catch( InvalidTransactionEx. e) {...}
```

Independent

```
try {
    Circle c = ...;
    c.setRadius(...);
}
catch( NegRadiusEx. e) {...}
```


String getRatingReport() {
String s = ... ;
return s;
}
① you don't necessarily have to have an attribute: String ratingReport;
② you can generate a string to return by going over some array and concatenate parts from that array.

Consider the following class:

```
public class Point {
  private double x;
  private double y;
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }
  public void moveUp(double units) {
    this.y = this.y + units;
  }
  public double getDistanceFromOrigin() {
    return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2));
  }
}
```

Now say we have the following variable declared and initialized:

```
Point p = new Point(3.4, 5.7);
```

From the following independent lines of code, chose those which compile (i.e., without any syntax or type error).

- [ ] a. System.out.println(p.moveUp(24.8));
- [ ] b. p.moveUp(24.8);
- [ ] c. int dist = p.getDistanceFromOrigin();
- [ ] d. double dist = p.moveUp(24.8);
- [ ] e. System.out.println(p.getDistanceFromOrigin());
- [ ] f. p.getDistanceFromOrigin();
- [ ] g. double dist = p.getDistanceFromOrigin();