Lecture 8 - Oct. 1

Exceptions

CoS Req.: Exec. Flow vs. Call Stack
To Handle or Not to Handle (V2 - V4)
More Examples on Exceptions

Announcements/Reminders

- Written Test 1 tomorrow (Wednesday)
- WrittenTest1 review session recording released
- Lab1 due this Friday
- Mockup Programming Test grading tests released
- Mockup Programming Test feedback to be released

Catch-or-Specify Requirement: Execution Flows

Normal Flow of Execution

```
... /* before, ouside try-catch block */
try {
    o.m(...); /* may throw SomeException */
    ... / * rest of try-block */
}
catch (SomeException se) {
    ... /* rest of catch-block */
}
.. /* after, ouside try-catch block */
```

When the exception does not occur

Abnormal Flow of Execution

```
... /* before, ouside try-catch block */
try {
    o.m(...); /* may throw SomeException */
    ... / * rest of try-block */
}
catch (SomeException se) {
    .. /* rest of catch-block */
}
. /* after, ouside try-catch block */
```

When the exception occurs

Catch-or-Specify Requirement: Call Stack

n method calls

Q1. Origin of Exception:

$C_1.m_1$

Q2. Methods subject to CoS Req:

$C_1.m_1 \sim C_{i-1}.m_{i-1}$

Q3. Methods free from CoS Req:

$C_{i+1}.m_{i+1}$

Q4. Extreme Case 1 (exception handled earliest):

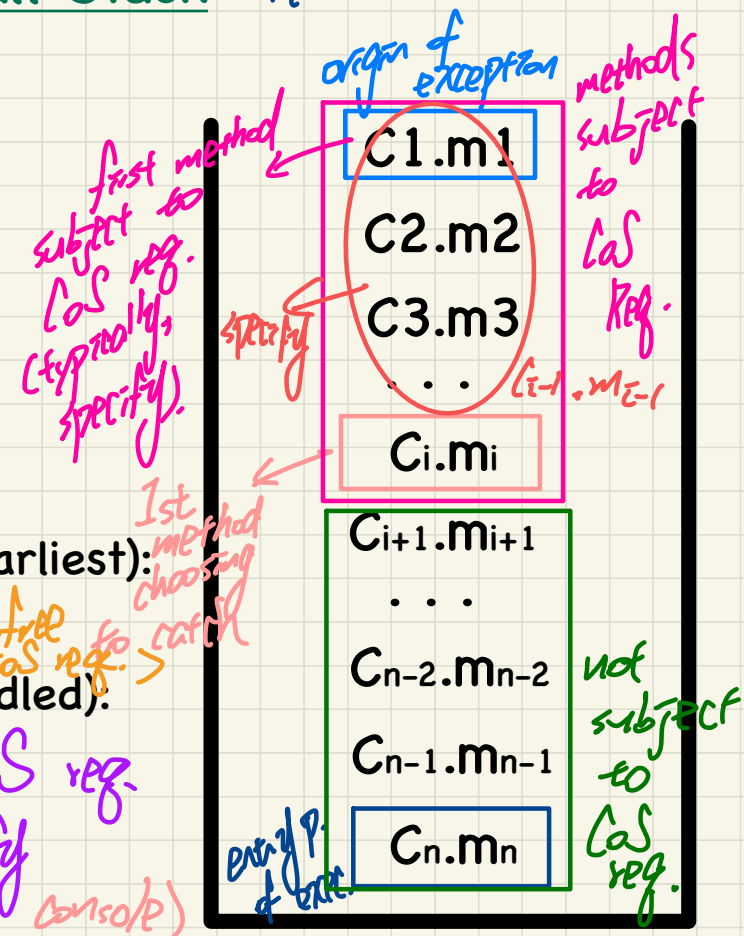
handled in $C_2.m_2$ ($C_3.m_3 \sim C_n.m_n$ free from CoS req.)

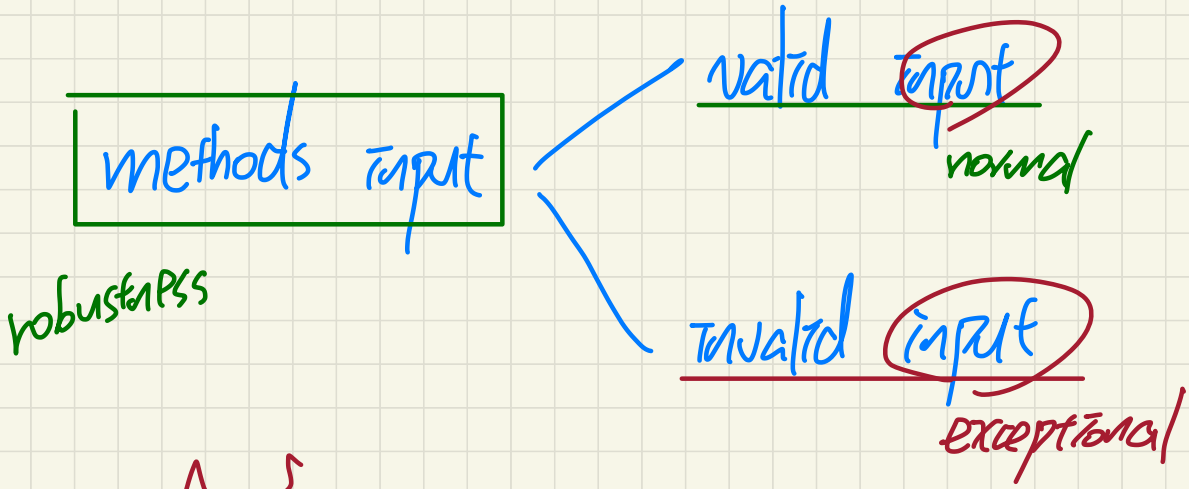
Q5. Extreme Case 2 (exception never handled):

$C_1.m_1 \sim C_n.m_n$ subject to CoS req.

but all chose to specify

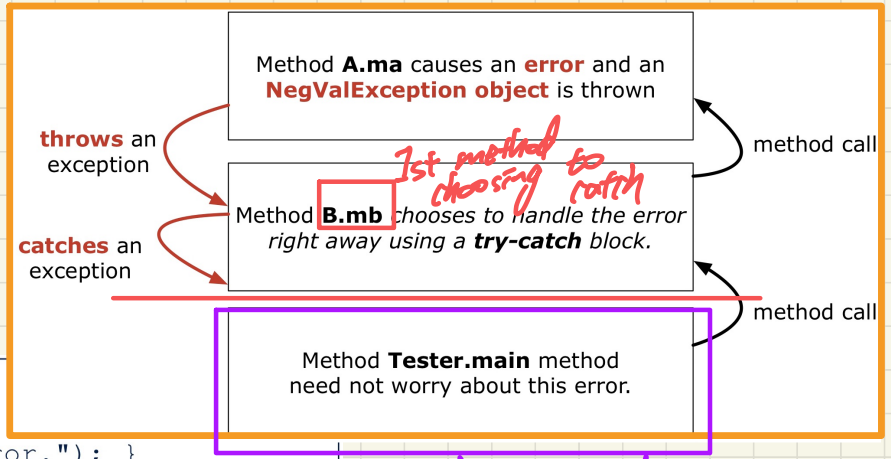
(exceptions thrown to console)




methods input
valid input
normal
invalid input
exceptional
robustness
{
m1(...){
o.m2(x)
}
}

Version 1:

Handle the Exception in B.mb



no longer subject to LoS req.

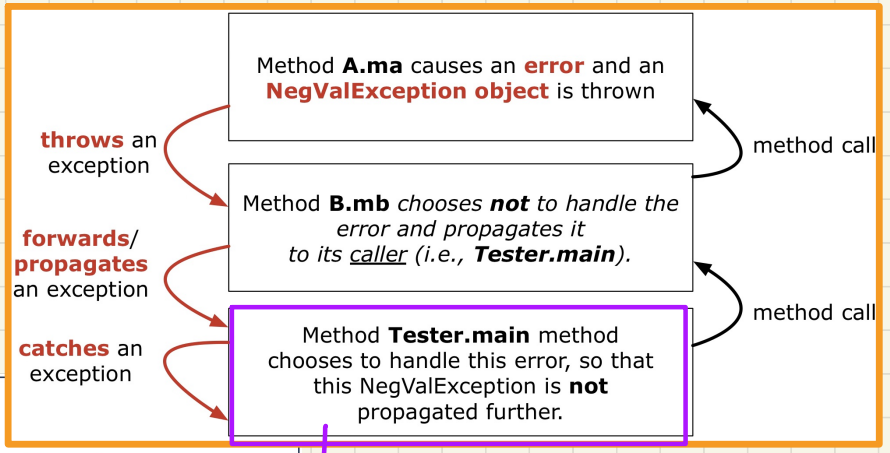
```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) {
    A oa = new A();
    try { oa.ma(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i); /* Error, if any, would have been handled in B.mb.
  } }
```

Version 2:

Handle the Exception in Tester.main




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

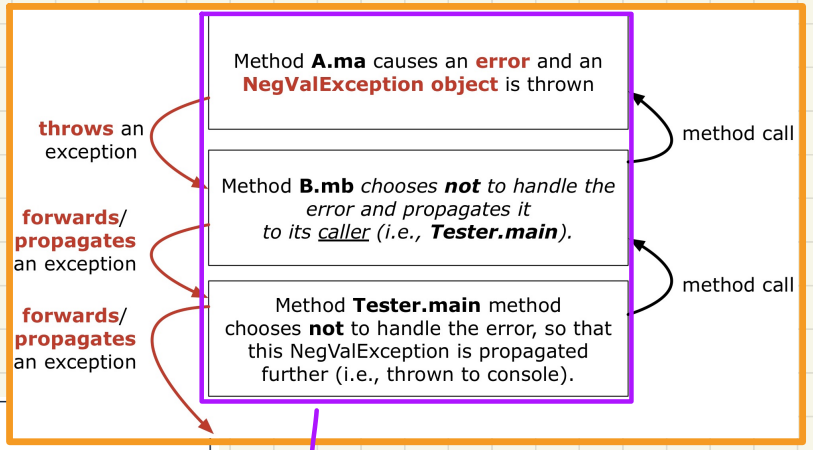
```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
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. */ }
  } }
```

State its caller has not handled the except; Tester.main stmt subject to CoS Req.

Version 3:

Handle in Neither Classes on Call Stack




```
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) throws NegValException {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i);
  } }
```

all methods subject to CnS Reqs., and all opt to specify.

Error Handling via Exceptions: Circles (Version 1)

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

Test Case 1:

User enters 10

Test Case 2:

User enters -5

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

caller of this method subject to C.o.S Req

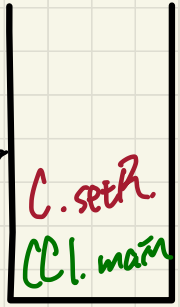
```
class CircleCalculator1 {
  public static void main(String[] args) {
    Circle c = new Circle();
    try {
      c.setRadius( 10 );
      double area = c.getArea();
      System.out.println("Area: " + area);
    }
    catch(InvalidRadiusException e) {
      System.out.println(e);
    }
  }
}
```

r == 10

Caller?

Callee?

call stack



Error Handling via Exceptions: Circles (Version 2)

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

Test Case:

User enters -5

Then user enters 10

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

```
public class CircleCalculator2 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean inputRadiusIsValid = false;
    while (!inputRadiusIsValid) {
      System.out.println("Enter a radius:");
      double r = input.nextDouble();
      Circle c = new Circle();
      try {
        c.setRadius(r);
        inputRadiusIsValid = true;
        System.out.print("Circle with radius " + r);
        System.out.println(" has area: "+ c.getArea()); }
      catch(InvalidRadiusException e) { System.out.print("Try again!"); }
    } } }
```

Error Handling via Exceptions: Banks

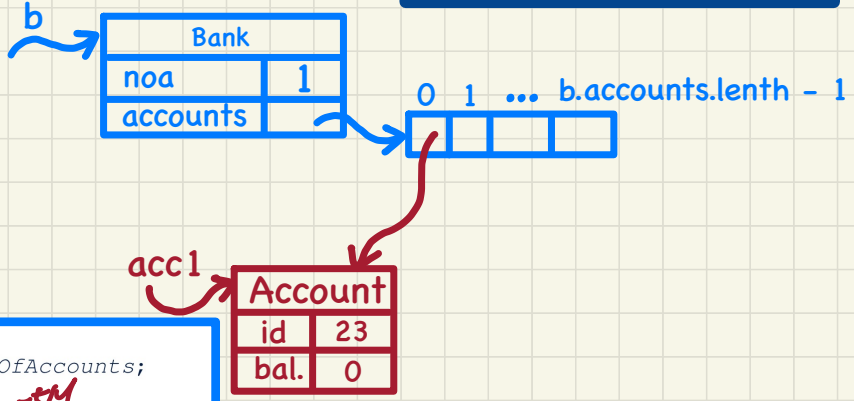
Test Case:
User enters -5000000

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

```
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; }
  }
}
```

```
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);
      }
    } /* end for */ }
```

```
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

accounts[i].withdraw(a)

Account[]

call stack

A.w
B.w
BA.main

More Example: Multiple Catch Blocks

Assume: customized exceptions only → order of their catch blocks does not matter.

```
1 double r = ...;
2 double a = ...;
3 try{
4   Bank b = new Bank();
5   b.addAccount(new Account(34));
6   b.deposit(34, 100);
7   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) {
9   System.out.println(r + " is not a valid transaction value.");
10  e.printStackTrace();
  }
```

→ may throw ITE

→ NPE may be thrown

Test Case 1:

a: -5000000

r: 23

Test Case 2:

a: 100

r: -5