Monday September 17

Lecture 4

Lab 1 part 2

2D arrays

nested loops

Error Handling with Console Messages: Circles

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) {
    if ( r < 0 ) { System.out.println( "Invalid radius." ); }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

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

this line should not be continued.

Error Handling with Console Messages: Call Chain

```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if (a < 0) { System.out.println("Invalid deposit."); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if (a < 0 || balance - a < 0) {
      System.out.println("Invalid withdraw."); }
    else { balance -= a; }
  }
}
```

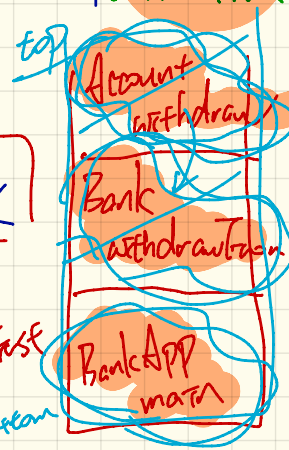
```
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdrawFrom(int id, double a) {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id) {
        accounts[i].withdraw(a);
      }
    } /* end for */
  } /* end withdraw */
}
```

```
class BankApplication {
  pubic static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a);
  }
```

Run as J. A.

user input

context class	caller	caller
Account	Account withdraw	
Bank	Bank withdrawFrom	Account withdraw
BankApp	main	Bank withdrawFrom



Stack

LIFO

Last In First Out

Circle Class with Exceptions (Example 1)

```
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; }
}
```

this part of the method signature becomes info to inform the potential caller of setRadius

exception object

```
IRE e = new IRE(": ..");
throw e;
```

```
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);
    }
  }
}
```

throw new IRE

accessor
return
normal

method
throw
abnormal

Enter radius:
-10
Invalid radius, try again:
-2
In ——— , t a. :
10
314

Circle Class with Exceptions (Example 2)

```
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; }
}
```

catch (IRE e) {
  e
  Syste
}

Case 1
User enters 10

Case 2
User enters -5

```
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); /* Error if r from the user is negative */
                inputRadiusIsValid = true;
                System.out.print("Circle with radius " + r);
                System.out.println(" has area: "+ c.getArea());
            }
            catch(InvalidRadiusException e) {
                System.out.println("Radius " + r + " is invalid, try again!");
            }
        }
        input.close();
    }
}
```

end of while

Bank Example with Exceptions

NAE: NegativeAmountException
ATLE: AmountTooLargeException

```
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; }
  }
}
```

throws NAE, ATLE

```
if(a<0){
  throw new NAE("neg. a.");
}
else if(balance - a < 0){
  throw new ATLE("too lar.");
```

```
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 */ } /* end withdraw */ }
```

throws NAE, ATLE;

```
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); } } }
```

NAE, ATLE

```
catch(NAE e){
  ...
}
catch(ATLE e){
  ...
}
```

To Handle or Not To Handle : VI

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

NVE is handle here
there's no need:
1. no need to "throws NVE" for mb
2. no try-catch in Tester.main when calling mb

```
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. */
  } }
```

To Handle or Not To Handle : V2

```
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);
  } }
```

no try-catch block

no need to write "throws NVE"

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

To Handle or Not To Handle : V3

```
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);
  } }
```

no try-catch

```
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);
  }
```

no try-catch

Integer.parseInt("256");

↳ 256

Integer.parseInt("two");

↳ NFE