CST256: Object Oriented Programming

UNIT - III

Kanak Kalyani

Exception Handling

```
class Exc0 {
public static void main(String args[]) {
int d = 0;
int a = 42 / d;

}
}
```

Stack Trace

Output:

java.lang.ArithmeticException: / by zero

at Exc0.main(Exc0.java:4)

```
class Exc1 {
static void subroutine(){
int d = 0;
int a = 10 / d;
}
public static void main(String args[]) {
Exc1.subroutine();
}
}
```

Output

```
java.lang.ArithmeticException: / by zero
at Exc1.subroutine(Exc1.java:4)
at Exc1.main(Exc1.java:7)
```

Exceptions Handling in Java

5 Keywords which help in exception handling in Java

- try
- catch
- finally
- throw
- throws

Using try and catch

```
class Exc2 {
public static void main(String args[]) {
    int d, a;
    try { // monitor a block of code.
        d = 0;
        a = 42 / d;
    System.out.println("This will not be printed.");
    }
    catch (ArithmeticException e) { // catch divide-by-zero error
        System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
    }
}
```

Output
Division by zero.
After catch statement.

Multiple catch Clauses

```
class MultiCatch {
    public static void main(String args[]) {
        Scanner sc=new Scanner(System.in);
        try {
            int a = sc.nextInt();
            System.out.println("a = " + a);
            int b = 42 / a;
            int c[] = { 1 };
            c[42]       = 99;
        }
        catch(ArithmeticException e) {
            System.out.println("Divide by 0: " + e);
        }
        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bound: " + e);
            }
        System.out.println("After try/catch blocks.");
    }
}
```

Nested try Statements

```
class MethNestTry {
  static void nesttry(int a) {
     try {
     int a = sc.nextInt();
     int b = 42 / a;
     System.out.println("a = " + a);
     try { // nested try block
     if(a==1)
     a = a/(a-a); // division by zero
     if(a==2) {
     int c[] = { 1 };
     c[42] = 99; // generate an out-of-bounds
     exception
     }}

catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index out-of-bounds:
" + e);}
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
}

 public static void main(String args[]) {
Scanner sc=new Scanner(System.in);
 try {
int a = sc.nextInt();
int b = 42 / a;
System.out.println("a = " + a);
nesttry(a);
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
}
```

Finally Keyword

```
public class ExcepTest{
public static void main(String args[]){
    int a[]=new int[2];
    try{
        System.out.println("Access element three :"+ a[3]);
    }catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Exception thrown  :"+ e);
    }
    finally{
        a[0]=6;
        System.out.println("First element value: "+a[0]);
        System.out.println("The finally statement is executed");
          }
}
}
```

Exception thrown :java.lang.ArrayIndexOutOfBoundsException:
Index 3 out of bounds for length 2
First element value: 6
The finally statement is executed

finally keyword

```
class FinallyDemo {
static void procA() {
try {
      System.out.println("inside procA");
      throw new RuntimeException("demo");
     }
finally {
      System.out.println("procA's finally");
    }
}

static void procB() {
try {
      System.out.println("inside procB");
return;
}
finally {
      System.out.println("procB's finally");
}
}
```

```
static void procC() {
try {
      System.out.println("inside procC");
}
 finally {
      System.out.println("procC's finally");
}
public static void main(String args[]) {
try {
      procA();
}
 catch (Exception e) {
      System.out.println("Exception caught");
}
procB();
procC();
} }
```

inside procA
procA's finally
Exception caught
inside procB
procB's finally
inside procC
procC's finally

throw keyword

```
class ThrowDemo {
 static void demoproc() {
try {
      throw new NullPointerException("demo");
 }
catch(NullPointerException e) {
 System.out.println("Caught inside demoproc.");
throw e; // rethrow the exception
}
}//demoprac
 public static void main(String args[]) {
try {
 demoproc();
}
catch(NullPointerException e) {
                System.out.println("Recaught: " + e);
try{
                throw e;
}
catch(NullPointerException e){
 System.out.println("Recaught: " + e);   }  }
}// main
}//class
```

Output
Caught inside demoproc.
Recaught: java.lang.NullPointerException: demo

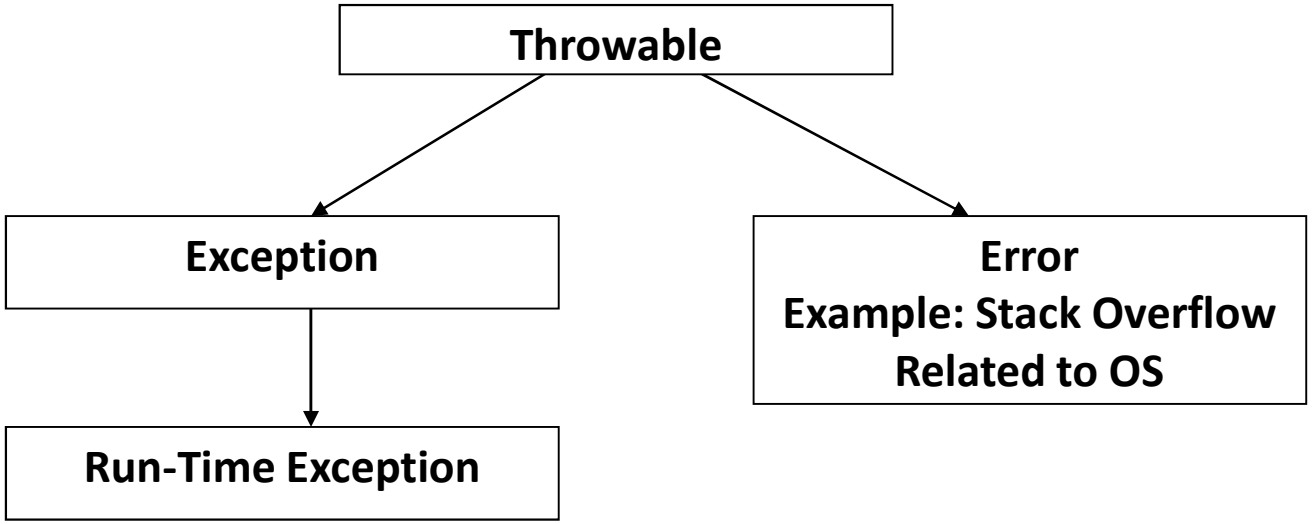
throws

type method-name(*parameter-list*) throws *exception-list*
{
// body of method
}

```
class ThrowsDemo {
static void throwOne() throws IllegalAccessException
 {
System.out.println("Inside throwOne.");
Throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
throwOne();
} catch (IllegalAccessException e) {
System.out.println("Caught " + e);
}
}
```

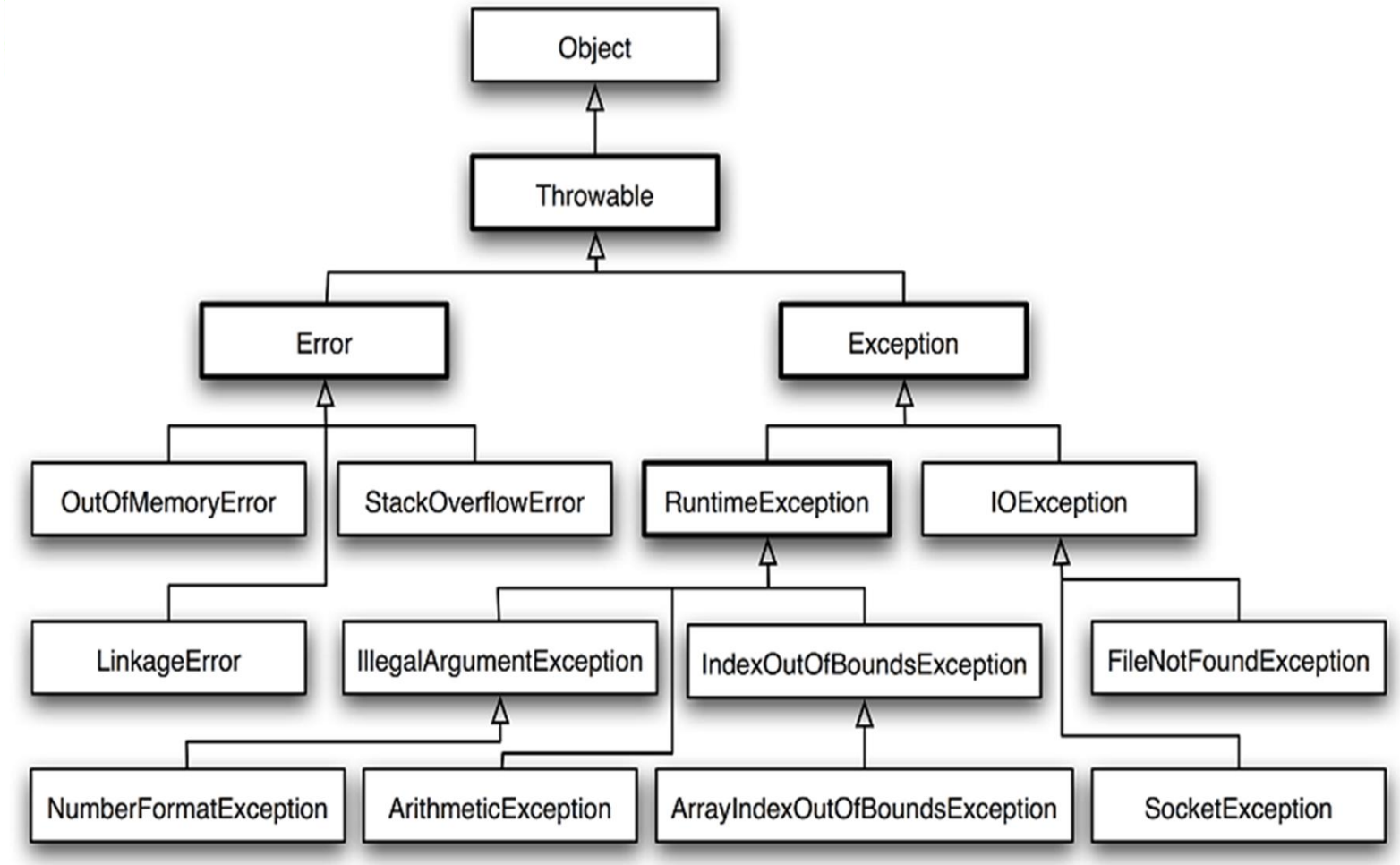
Output
inside throwOne
caught java.lang.IllegalAccessException: demo

Exception Hierarchy



```
class unreachablecatch{
public static void main(String args[]) {
try {
int a = 0;
int b = 42 / a;
} catch(Exception e) {
System.out.println("Generic Exception catch.");
}
catch(ArithmeticException e) { // ERROR - unreachable
System.out.println("This is never reached.");
}
```

Exception Hierarchy



User Defined Exception

```
class MyException extends Exception {
private int detail;
MyException(int a) {
        detail = a;
}
public String toString() {
        return "MyException[" + detail + "]";
}
class ExceptionDemo {
        static void compute(int a) throws MyException  {
        System.out.println("Called compute(" + a + ")");
        if(a > 10)
                throw new MyException(a);
        System.out.println("Normal exit");
        }
}
public static void main(String args[]) {
try {
        compute(1);
        compute(20);
} catch (MyException e) {
System.out.println("Caught " + e);
}}
```

Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]

```
public class InsufficientFundsException extends
Exception {
  private double amount;
  public InsufficientFundsException(double
amount)  {
    this.amount = amount;
  }
  public double getAmount()  {
    return amount;
  }}
public class CheckingAccount{
  private double balance;
  public CheckingAccount(double amt){
    this.balance = amt;
  }
  public void deposit(double amount) {
    balance += amount;
  }
  public void withdraw(double amount) throws
  InsufficientFundsException{
    if(amount <= balance){
      balance -= amount;
    }
    else {
      double needs = amount - balance;
      throw new InsufficientFundsException(needs);
    }  }
  public double getBalance()  {
    return balance;
  }
  public int getNumber()  {
    return number;
  }}
public class BankDemo{
  public static void main(String [] args){
    CheckingAccount c = new
CheckingAccount(101);
    System.out.println("Depositing $500...");
    c.deposit(500.00);
    try {
      System.out.println("\nWithdrawing $100...");
      c.withdraw(100.00);
      System.out.println("\nWithdrawing $600...");
      c.withdraw(600.00);
    }catch(InsufficientFundsException e){
      System.out.println("Sorry, but you are short $"
                         + e.getAmount());
      e.printStackTrace();
    }  }}
```

Do it Yourself

Create a function to take input from user 10 positive numbers. If the number entered is less than 0 throw an Arithmetic exception. The exception should be handled in main().

Create a class Employee which stores UserID and Password. The password is valid if it contains a UpperCase letter, a LowerCase letter, a number and a special character(@,$,&,*) and length should be minimum of 8 characters.

If the password is not valid then throw user defined exception **InvalidPasswordException**.