Exceptions

- **What if we do not handle Exceptions**
- **Handling Exceptions**
- **Recovering from Exceptions**
- **Exception Hierarchy**
- **Propagation of Exceptions**
- **The clause finally**
- **Re-throwing Exceptions**
- **Writing our own exceptions**

1

Exceptions

Recall our 2nd week program.

This method is not prepared to handle it!

```
import java.io.*;
public class AddingInts {
   public static void main (String[] args)
                                    throws IOException {
      BufferedReader stdin = new BufferedReader
         (new InputStreamReader (System.in));
      String string1, string2;
      int num1, num2, sum;
      System.out.println ("Input an integer");
      string1 = stdin.readLine();
      num1 = Integer.parseInt (string1);

      System.out.println ("Input another integer");
      string2 = stdin.readLine();
      num2 = Integer.parseInt (string2);
      sum = num1 + num2;
      System.out.print("The sum is: " + sum);
   }
}
```

may throw an IOException

may throw a NumberFormatException

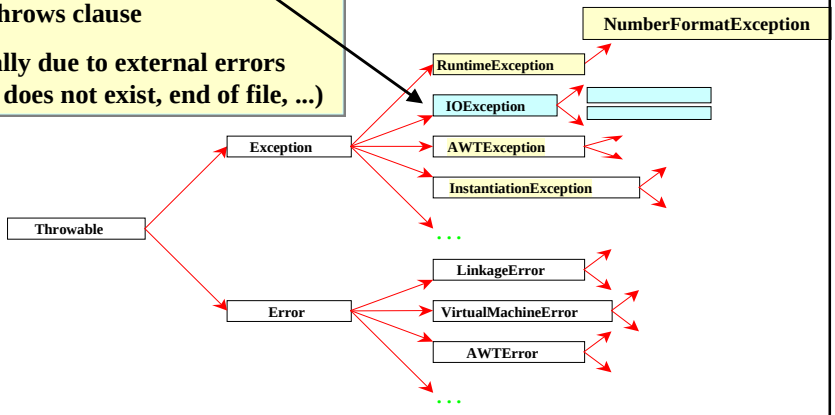
How come there is no throws NumberFormatException ?

Exception classes Hierarchy

Checked Exception

must be caught or listed in the throws clause

Usually due to external errors (File does not exist, end of file, ...)



3

We did not handle the exceptions

When the main method does not handle the exception the program aborts.

Considered Poor programming practice

Next program catches both type of exceptions but does not attempt to recover from them.

4

```
import java.io.*;
class AddingInts {
  public static void main (String[] args)  {
    BufferedReader stdin = new BufferedReader
                  (new InputStreamReader (System.in));
    String string1, string2;
    int num1 = 0, num2 = 0, sum = 0;

    try  {
      System.out.println ("Input an integer number");
      string1 = stdin.readLine();
      num1 = Integer.parseInt (string1);

      System.out.println ("Input an integer number");
      string2 = stdin.readLine();
      num2 = Integer.parseInt (string2);
    }
    catch (Exception e) {
      System.err.println("Problem in input.  Exiting ..");
      System.err.println(e);
      System.exit(1);
    }
    sum = num1 + num2;
    System.out.println("The sum is: " + sum);
  }
}
```

Catches any exception that belong Exception or one of its subclasses.

5

Refinement 1 … Catches the exceptions separately

```
try  {
  System.out.println ("Input an integer number");
  string1 = stdin.readLine();
  num1 = Integer.parseInt (string1);
  System.out.println ("Input an integer number");
  string2 = stdin.readLine();
  num2 = Integer.parseInt (string2);
}
catch (NumberFormatException nfe) { // wrong input
  System.err.println("Invalid input format.  " + "Exiting ...");
  System.err.println(nfe);
  System.exit(1);
}
catch (IOException ioe) {
  System.err.println("Problem in input.  Exiting ...");
  System.err.println(ioe);
  System.exit(1);
}
```

Subclasses of Exception

Still unable to recover from the error ! Next Refinement attempts it.

6

Refinement 2 … Repeat until user enters valid numbers

Repeat until both numbers are valid

```
boolean valid = false;
do {
   try {
      System.out.println ("Input an integer number");
      string1 = stdin.readLine();
      num1 = Integer.parseInt (string1);

      System.out.println ("Input an integer number");
      string2 = stdin.readLine();
      num2 = Integer.parseInt (string2);
      valid = true;
   }
   catch (NumberFormatException nfe) {
      System.err.println("Invalid input: try again");
      System.err.println(nfe);
   }
   catch (IOException ioe) {
      System.err.println("Problem in input.  Exiting ..");
      System.err.println(ioe);
      System.exit(1);
   }
} while (!valid);
```

Initially set to false

All okay up to now - (no exceptions thrown) set valid to true

Repeats loop until valid is set to true

Propagation of Exceptions

What happens If an exception is thrown and there is no `catch` clause ?

The exception propagates back to the caller of the method

The method below will throw an ArithmeticException if den is passed the value of 0. As it is not caught in the same method it is passed to the calling method and handled there.

```
public static int divide(int num, int den) throws ArithmeticException  {
            return num/den;
}
```

8

```
import java.io.*;
public class CheckException{
  public static void main (String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        int i, j, ans=1;
        boolean okay = false;
        System.out.print("Enter numerator : ");
        i = console.readInt();
        do {
          try {   System.out.print(”Enter demoninator : ");
                  j = console.readInt();
                  ans = divide(i,j);
                  okay = true;
          }
          catch(ArithmeticException e) {
                  e.printStackTrace();
                  System.out.println("Value for denom cannot be 0");
          }
        } while ( !okay );
        System.out.println("Answer is " + ans);
  }
  public static int divide(int num, int den) throws ArithmeticException  {
          return num/den;
  }
}
```

9

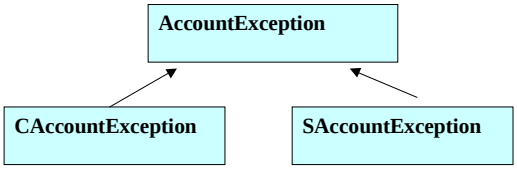
Sample Input/Output

```
Enter numerator          : 8
Enter denominator : 4
Answer is 4
```

```
Enter numerator          :  8
Enter denominator :  0
java.lang.ArithmeticException: / by zero
        at CheckException.main
value for denom cannot be 0
Enter denominator
2
Answer is 4
```

Accounts Example

- **As a constructor cannot return a value, it may throw an exception. For example the Account constructor may throw an exception if initial balance passed is negative.**
- **SAccount may throw an exception if min-amount > initial-balance**
- **To cater for these we may create our own exception classes. AccountException with subclasses SAccountException and CAccountException.**



11

```
Account acc;
System.out.printn("Specify Account type : ");
String type = console.readLine();
try {
if ( type.charAt(0) == 'A')  acc = new Account(….)
else if (type.charAt(0) == 'C')  acc = new CAccount(….)
else if ( type.charAt(0) == 'S')  acc = new SAccount(….)
}
catch (AccountException ae) { ….}
catch (CAccountException cae) { ….}
catch (SAccountException sae) { ….}
```

All exceptions will be caught by this as it is the superclass - hence rearrange them

```
catch (CAccountException cae) { ….}
catch (SAccountException sae) { ….}
catch (AccountException ae) { ….}
```

Catch the more specialized classes first.

(handled correctly)

12

The finally construct

You may want to take some action whether or not exception is thrown (such as closing a file).

The `finally` is used to handle this situation.

In the code below statements B and D may not be reached but statement, C will always be executed.

```
try{  A; // may throw ExType1 or ExType2
      B; //;
}
catch(ExType1 e){
    handling e;
}
finally{
   C; // finalStatements;
}
D ;
```

13

Rethrowing exceptions

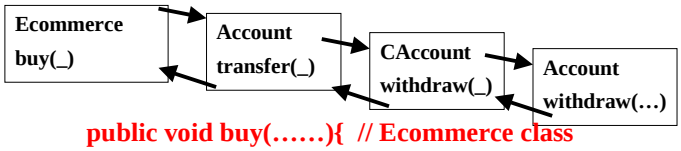
When an exception occurs the enclosing method exits immediately, unless the exception is caught.

If we need to perform some tasks before exiting, we can catch it and then *rethrow* it again.

```
try
{  ….. .
}
catch(TheException e)
{
     perform operations before exits;
     throw e;
}
```

14

Why throw exceptions ?




```
public void buy(……){  // Ecommerce class
  do {     // may prompt user to enter different values
           …
           if (c1.transfer(c2,amount) == true) {

public boolean transfer (……){  //  Account class
           if (withdraw(..) == true) {

public boolean withdraw (……){  //  CAccount class
           if (super.withdraw(..) == true) {

public boolean withdraw (……){  //  Account class
           if (balance > amt )  {
                    …
                    return true;
```

15

Using the Exception mechanism

```
public void buy(……){  // handles exception
 do {      // may prompt user to enter different values
           try { ...
                 c1.transfer(c2,100)
           catch(WithdrawException we) {  ……..  }

public void transfer (……) throws WithDrawException {
            withdraw();

public void withdraw (……) throws WithDrawException {
super.withdraw();

public void withdraw (……) throws WithDrawException {
           if (balance > amt )  {
                    …
           }
           else throw new WithdrawException(…);
```

Propagating the WithDrawException

16

Our Own Exceptions

- **The code below defines our own exception class which is thrown when a user attempts to withdraw a –ve *amount* or when the *amount* exceeds the avilable balance.**
- **This class has *instance variables* for storing information about the error condition.**

```
class WithdrawException extends Exception
{ public String getCause() {        return cause; }
  public double maxAvailable() { return maxAvailable;}
  public WithdrawException(String cause) { this.cause = cause; }
  public WithdrawException(String cause,
          double maxAvailable) {
          this.cause = cause;
          this.maxAvailable = maxAvailable;
  }
  private          String cause;
  private          double maxAvailable; //max. withdrawable
}
```

17

- **Our `Account` class withdraw can now be redefined to throw this exception whenever any condition is detected.**
- **Notice the exception object is storing an *error message* and the *maximum amount* that can be withdrawn.**
- **This information may be used to recover from the error.**

```
// if insufficient funds WithdrawException will be thrown
public void withdraw(double amount) throws WithdrawException {
    if (amount < 0 )
        throw new  WithdrawException("Negative Amount");
    if (balance <  amount)
        throw new WithdrawException("Amount Not Available", balance);
    balance = balance - amount;
}
```

18

Recovering ...

The program segment next shows how we can recover from this exceptional condition using the `WithdrawException` object caught.

19

```
ConsoleReader console = new ConsoleReader(System.in);
double amount;
boolean amountOK = false;
System.out.println("Enter Amount : ");
amount = console.readDouble();
while (!amountOK) {
        try {       cDad.withdraw(amount);
                    amountOK = true;
        }
        catch (WithdrawException we) {
           System.out.println(we.getCause());
           if (we.getCause().compareTo("Amount Not Available") == 0) {
              System.out.println("Wish to borrow max amount ?  (Y/N) ");
              if (console.readLine().charAt(0) == 'Y' ) {
                          amount = we.maxAvailable();
                          continue;
              }
           }
        }
        if (!amountOK) {
                    System.out.println("ReEnter Amount : ");
                    amount = console.readDouble();
        }
};
```

20