

## Error Handling

## Exceptions (I)

- Allow to jump from a point (throw) to another (catch) anywhere in the program
  - Independent programming of exception throwing and catching
- Some information is sent along with the jump: an exception
  - An exception is an object of a specific class – it holds information about the error condition that occurred
  - Each *catch* captures a type of exception, and ignores the rest
- Usefulness: error handling/recovery

2

## Exceptions (II)

```
void f () throws NegativeAge {  
    ...  
    if (...) throw new NegativeAge (person, age);  
    ...  
}  
  
try {  
    ...  
    obj.f ();  
    ...  
}  
catch (NegativeAge ex) {  
    ...  
}
```

3

## Example

### 1. Throwing exceptions

```
class BankAccount {  
    ...  
    boolean blocked;  
    ...  
    void withdraw (long amount) throws BalanceUnderflow,  
        BlockedAccount {  
        if (blocked)  
            throw new BlockedAccount (number);  
        else if (amount > balance)  
            throw new BalanceUnderflow (number, balance);  
        else balance -= amount;  
    }  
}
```

4

### 2. Defining exception classes

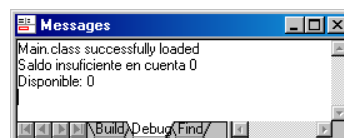
```
class BalanceUnderflow extends Exception {  
    long number, balance;  
    BalanceUnderflow (long num, long s) {  
        number = num; balance = s;  
    }  
    public String toString () {  
        return "Insufficient balance in account " + number  
            + "\nAvailable: " + balance;  
    }  
}  
  
class BlockedAccount extends Exception {  
    long number;  
    BlockedAccount (long num) { number = num; }  
    public String toString () {  
        return "Account nr. " + number + " is blocked";  
    }  
}
```

5

### 3. Catching and handling exceptions

```
static public void main (String args[]) {  
    try {  
        new BankAccount () .withdraw (100000);  
    }  
    catch (BalanceUnderflow ex) {  
        System.out.println (excep.toString ());  
    }  
    catch (BlockedAccount ex) {  
        System.out.println (excep.toString ());  
    }  
}
```

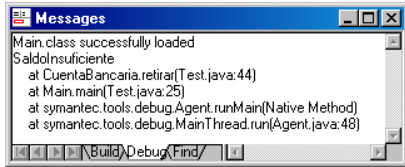
The first catch with compatible type is executed



6

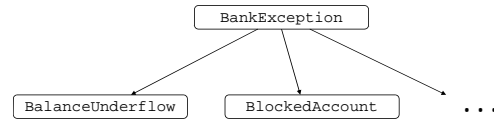
#### 4. ¿What if an exception is not captured?

```
static public void main (String args[])
    throws BlockedAccount, BalanceUnderflow {
    BankAccount account = new BankAccount ();
    account.balance = 1000;
    account.withdraw (2000);
}
```



7

#### Exception Hierarchies



##### Generic handling

```
catch (BankException obj) {
    ...
}
```

##### Specific handling

```
catch (BalanceUnderflow obj) {
    ...
}
catch (BlockedAccount obj) {
    ...
}
```

8

#### Exceptions are Part of an Object's Interface

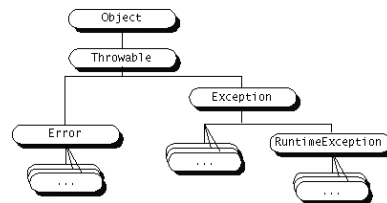
- If a method lets an exception pass, it must be declared in the method header
- Only `Errors` and `RuntimeExceptions` do not need to be declared
- An overridden method cannot declare more exceptions than those declared in the parent's definition (or subclasses thereof)
- Therefore, if an overridden method throws an exception not declared in the parent class, it is mandatory to catch it even if nothing is done

```
class X extends Applet {
    public void start () { // start inherited from Applet
        try { ... } (catch IOException e) { /* empty */ }
    }
}
```

9

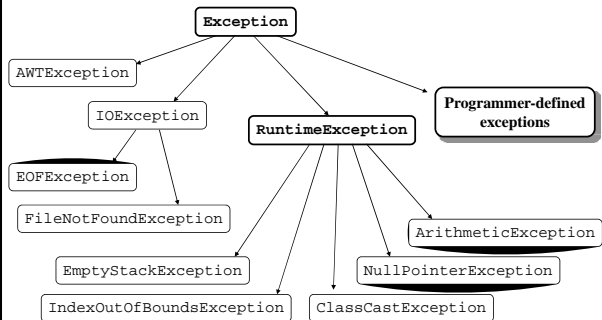
#### Predefined Exceptions

- Java runtime errors are produced as exceptions
- Runtime exceptions: null pointers, array bounds, arithmetic, etc.
  - They are not checked
  - In general, they should not be subclassed
- Error: link errors, memory overflow, etc. (severe errors)
  - In general, should be neither caught nor subclassed



10

#### Some Predefined Exceptions



11

#### Methods of the Throwable class

- `Throwable(String)` Assigns a message to the exception
  - `getMessage()` Returns the message of the exception
  - `toString()` Returns a string with the name of the exception class, plus the exception message
  - `printStackTrace()` Prints the runtime stack trace to standard error
- When an exception is not caught anywhere, the program stops and `printStackTrace()` is called
- To take advantage of the methods of `Throwable`, define a constructor:
 

```
BlockedAccount (long num) {
                super ("Account nr. " + num + " is blocked");
                number = num;
            }
```

12

### Advantages of Exceptions

- Separate error handling from the rest of the program code
  - Get rid of error codes
  - Get rid of explicit control flow alteration
- Error propagation through method call stack
  - Avoid returning error code values
  - Get rid of additional error status arguments
- Error type system, group types of errors
  - Exception class hierarchies
  - Handle errors at the desired level of specificity

13

### Without Error Handling

```
readFile {  
  open the file;  
  determine its size;  
  allocate that much memory;  
  read the file into memory;  
  close the file;  
}
```

14

### Error Handling without Exceptions

```
errorCodeType readFile {  
  initialize errorCode = 0;  
  open the file;  
  if (theFileIsOpen) {  
    determine the length of the file;  
    if (gotTheFileLength) {  
      allocate that much memory;  
      if (gotEnoughMemory) {  
        read the file into memory;  
        if (readFailed) errorCode = -1;  
      }  
      else errorCode = -2;  
    }  
    else errorCode = -3;  
    close the file;  
    if (theFileDintClose && errorCode == 0)  
      errorCode = -4;  
    else errorCode = errorCode & -4;  
  }  
  else errorCode = -5;  
  return errorCode;  
}
```

15

### Error Handling with Exceptions

```
readFile {  
  try {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
  }  
  catch (fileOpenFailed) { doSomething; }  
  catch (sizeDeterminationFailed) { doSomething; }  
  catch (memoryAllocationFailed) { doSomething; }  
  catch (readFailed) { doSomething; }  
  catch (fileCloseFailed) { doSomething; }  
}
```

16

### Without Error Handling

```
method1 {  
  call method2;  
}  
  
method2 {  
  call method3;  
}  
  
method3 {  
  call readFile;  
}
```

17

### Error Handling without Exceptions

```
method1 {  
  errorCodeType error;  
  error = call method2;  
  if (error) doErrorProcessing;  
  else proceed;  
}  
  
errorCodeType method2 {  
  errorCodeType error;  
  error = call method3;  
  if (error) return error;  
  else proceed;  
}  
  
errorCodeType method3 {  
  errorCodeType error;  
  error = call readFile;  
  if (error) return error;  
  else proceed;  
}
```

18

### With Exceptions

```
method1 {  
  try {  
    call method2;  
  }  
  catch (exception) {  
    doErrorProcessing;  
  }  
}  
  
method2 throws exception {  
  call method3;  
}  
  
method3 throws exception {  
  call readFile;  
}
```

19