G51PRG:
Introduction to Programming
Second semester
Lecture 8

Natasha Alechina
School of Computer Science & IT
`nza@cs.nott.ac.uk`

Previous lecture

- More about interfaces and collections
- test on polymorphism and inheritance
- answer to Book and Textbook exercise
- introduction to the next exercise

This lecture: exceptions

- what are exceptions for
- how to define your own exception
- how to get a method to throw an exception
- how to catch and handle exceptions

Exceptions you may have seen

```
ArrayIndexOutOfBoundsException
ClassCastException
NullPointerException
CloneNotSupportedException
```

- They all are classes which extend the Trowable class of java.lang.
- New user defined exceptions usually extend its subclass, Exception.

What are exceptions for

- Lots of things can go wrong during the execution of a program.
- Sometimes you can predict all of them and write lots of if-statements which deal with all possible problems. This may clatter the code.
- Sometimes you simply cannot predict all things which can go wrong.
- Exceptions are a clean way to check for errors without cluttering code. They also provide a mechanism to signal errors directly and handle them.

How exceptions work

- An exception is ***thrown*** when an error condition is encountered.
- If an exception is thrown, there are two possible scenarios:
  - some code is provided which says what to do with the exception (the exception is ***caught***). Then this code is executed, and the program continues.
  - the exception is not caught by the method which was active when the exception was thrown. It is passed to the method which called it, and so on, until it is caught by the default exception handler. Default exception handler terminates the program and prints some information about the exception and where it was thrown.

Exception class

- If you want to create a new exception (to signal a specific error which may occur during execution of your program) you extend Exception class.
- Constructors:

`Exception()` - constructs an Exception with no specified detail message.

`Exception(String s)` - constructs an Exception with the specified detail message.

- useful method:

`public String getMessage()` - returns the error message

Example: ItemNotFoundException

- Suppose you want to write an exception which is thrown by a method which searches an array for a certain item. If the item is found, its index in the array is returned; otherwise an exception is thrown.

```
class ItemNotFoundException extends
  Exception{
  public ItemNotFoundException(String s){
   super("No item \"" + s + "\" found");
  }
}
```

Throwing exceptions

- Exceptions which a method throws are as important as its return type.
- If a method throws an ***checked*** exception and does not catch it, it should be declared with a throws clause:

```
public static int search(Object[] arr,
  Object it)throws ItemNotFoundException
```

- all user defined exceptions are checked
- unchecked exceptions: e.g. **`ArrayIndexOutOfBoundsException`**

To throw an exception in the code

```
throw ExceptionObject;
```

For example:

```
Exception myException = new
  ArrayIndexOutOfBoundsException();
throw myException;
```

or

```
throw new
  ArrayIndexOutOfBoundsException();
```

Example

```
public static int search(Object[] arr,
Object it) throws ItemNotFoundException {

 for(int i = 0; i< array.length; i++){
  if(array[i].equals(it)) return i;
 }
throw new
  ItemNotFoundException(it.toString());
}
```

Dealing with exceptions

If you invoke a method which throws a checked exception, you have two choices:

- declare the exception in your throws clause (as above)
- catch the exception and handle it

Passing the exception on

First choice:

- declare the exception in your throws clause

e.g. **`search(Object[] arr, Object it) throws ItemNotFoundException`**

this means that any method which calls **`search()`** either has to handle this exception or also declare it.

If the exception is thrown and not handled it is eventually caught by the default exception handler and execution of the program stopped.

Example

```
public static void main (String[] args)
  throws ItemNotFoundException {
  Object[] array = new Object[2];
  array[0] = new String("string");
  array[1] = new Integer(345);
  int i = search(array, "STRING");
  System.out.println("Life goes on");
}
```

Exception will be thrown since “STRING” is not in the array.

Example continued

The result is: execution stops, error message and call stack is printed:

```
ItemNotFoundException: No item "STRING" found
        at java.lang.Throwable.(Compiled Code)
        at java.lang.Exception.(Compiled Code)
        at ItemNotFoundException.(Compiled Code)
        at Search.search(Compiled Code)
        at Search.main(Compiled Code)
```

Catching exceptions

Option two: catch the exception and handle it (specify what should be done if the exception is thrown).

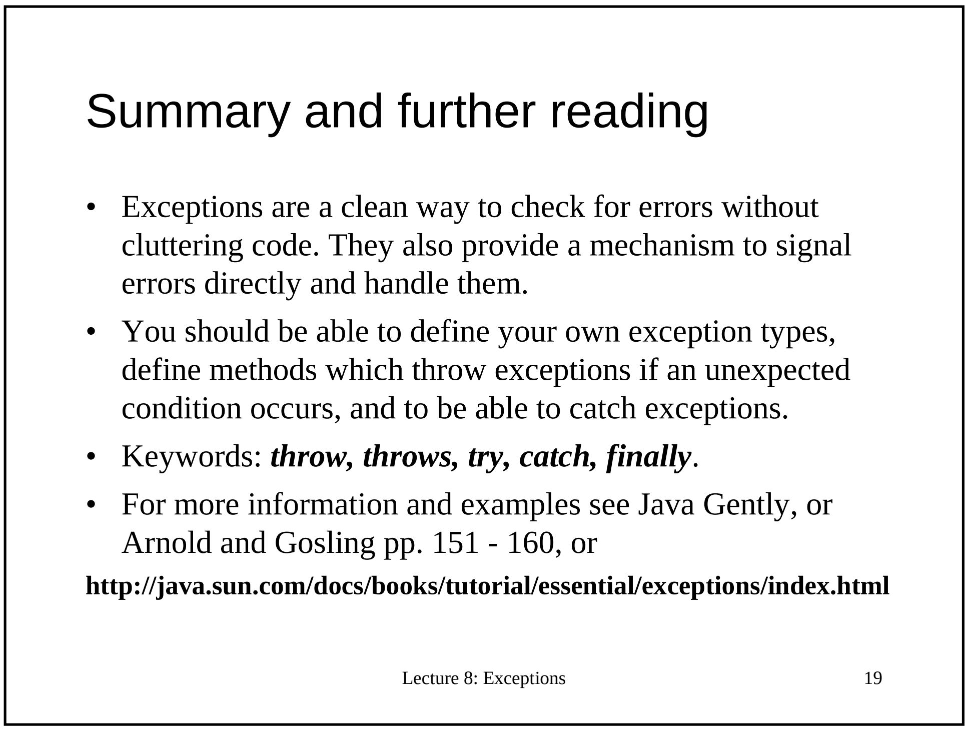
```
try {
  statements;
}
catch(exception_type1, identifier1){
  statements1;
}
catch(exception_type2, identifier2){
  statements2;
}
...
} finally {
  statementsN;
}
```

Try-catch-finally

- The try block is the "normal" code, for example a call to **`search().`**
- The catch statements are executed if an exception of appropriate type is thrown; there can be several catch blocks for different kinds of exceptions; they may in turn throw other exceptions.
- The finally block can be omitted; statements in finally block are executed always; usually some kind of emergency clean-up.

Example

```
public static void main(String[] args) {
 Object[] array = new Object[2];
 array[0] = new String("string");
 array[1] = new Integer(345);
 try {
  int i = search(array, "STRING");
 }
 catch(ItemNotFoundException e){
  System.out.println(e.getMessage());
 }
 System.out.println(“Got here”);
}
```



Summary and further reading

- Exceptions are a clean way to check for errors without cluttering code. They also provide a mechanism to signal errors directly and handle them.
- You should be able to define your own exception types, define methods which throw exceptions if an unexpected condition occurs, and to be able to catch exceptions.
- Keywords: ***throw, throws, try, catch, finally***.
- For more information and examples see Java Gently, or Arnold and Gosling pp. 151 - 160, or

http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html