

## Single vs. Multiple Inheritance

- o C# and Java only have single inheritance, although a class can implement as many interfaces (separated by commas) as it wants
- o In C# you can't really tell if something is an interface (in Java instead of extends interfaces use the keyword implements), so it's common to see interfaces start with a capital I. We can do this in class for clarity.

## An interface is a type

- o When you define an interface, you define a new reference data type
- o You can use interface names anywhere you can use other data type name.

## Interfaces shouldn't...

- o Interfaces shouldn't grow over time
  - o Every class using an interface would have to change if the interface changed
- o Interface can be extended by declaring a secondary interface – you can use inheritance with interfaces

## Exceptions

- o What happens when a method's precondition(s) is/are not met?
- o Often, an exception is “thrown”

## Exception Definition

- o An exception is an event that occurs during the normal flow of execution of a program that disrupts the normal flow of execution.
  - o Examples: a hard disk crash, array index out of range, etc.

## In Java or C#

- o An *exception* is an object, that is “thrown”. The thrown object is passed up through the method call stack until it is “caught” by an *exception handler*. If the exception is not caught, the program will terminate when the exception propagates out of the Main method.

## In C#

- o The class `Exception` is that top of the exception hierarchy. Under this class exists two different subclasses:
  - o `SystemException`
  - o `ApplicationException`

## Java and Checked Exceptions

- o Exceptions that are not subclasses of `RuntimeException` in Java are *checked* exceptions: They have to be caught or declared (listed in the method's throw clause). In other words: the application must somehow explicitly deal with the exception.
- o C# does not have checked exceptions. This note sheds some light on why:
  - o <http://discuss.develop.com/archives/wa.exe?A2=ind0011A&L=DOTNET&P=R32820>

## try/catch/finally Statements

- o the `try` block is wrapped around the statement that may cause an exception
- o the `catch` blocks following the `try` block are exception handlers. They contain code to handle the exception
- o The first handler (`catch` block) immediately following the `try` block whose parameter is compatible with the thrown object will be executed.
- o If there is no handler, the exception is propagated up the call stack.
- o There is an optional `finally` block that contains code that will always be executed.

## try/catch/finally Statement Syntax

```
try {
    statements
} catch (Exception1 e1) {
    statements
} catch (Exception2 e2) {
    statements
}
...
} catch (Exceptionk ek) {
    statements
} finally {
    statements
}
```

## The Bomb, the Basket and the Balloon

- o A framework for thinking about exceptions

## Handling Exceptions

- o [handout up on the web site]
- o Exception Propagation
- o Statement Syntax: `try/catch/finally`
- o Throwing exceptions
- o Creating your own exceptions

## Exceptions

- o Let's practice
- o Several example programs will be given out and it is your job to figure out what they do and to explain this to your classmates.

## Read

- o In the Java vs. C# handout
  - o Wish you were here
    - o Checked Exceptions
- o Discuss:
  - o When do you throw exceptions?
  - o Do you think C# should have checked exceptions?

## Designing with Exceptions

(obtained from <http://www.artima.com/designtechniques/desexcept.html>)

- o Benefits of exceptions
  - o Allow you to separate error handling code from normal code. This allows normal code to look less cluttered.
  - o It enables methods to "throw" exceptions, so that somebody else must deal with the particular error.
  - o Checked exceptions (in Java) force client programmers to deal with the potential exception.

## Testing Substring

```
using System;
public class TestSubstring {

    public static void Main( string[] args ) {
        string str = "I am a string";
        Console.WriteLine("An index that exists is: " +
            str.Substring(0, 1));
        Console.WriteLine("A negative index that doesn't exist returns: " +
            str.Substring(-1, 2) );
        Console.WriteLine("An index that is too large: " +
            str.Substring(0, 500) );
        Console.WriteLine("Backwards indices return: " +
            str.Substring(3, 0) );
    }
}
```

## Exceptions Indicate a Broken Contract

- o Example from the string.Substring() method:
  - o Precondition of Substring( int index, int length ): the index parameter passed must be between 0 up to and including the string.Length value
  - o Postcondition: The return value will be the substring from the starting index and through the length and the string itself will remain unchanged.
  - o If the precondition isn't met, Substring() throws System.ArgumentOutOfRangeException. This indicates a problem with the calling of the method.

## When to Throw Exceptions

- o If your method encounters an abnormal condition that it can't handle, it should throw an exception.

o What's an "abnormal condition"?

