C# Language & .NET Platform
12th Lecture

http://d3s.mff.cuni.cz/~jezek

Pavel Ježek

pavel.jezek@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Some of the slides are based on University of Linz .NET presentations.
© University of Linz, Institute for System Software, 2004
published under the Microsoft Curriculum License
(http://www.msdnaa.net/curriculum/license_curriculum.aspx)

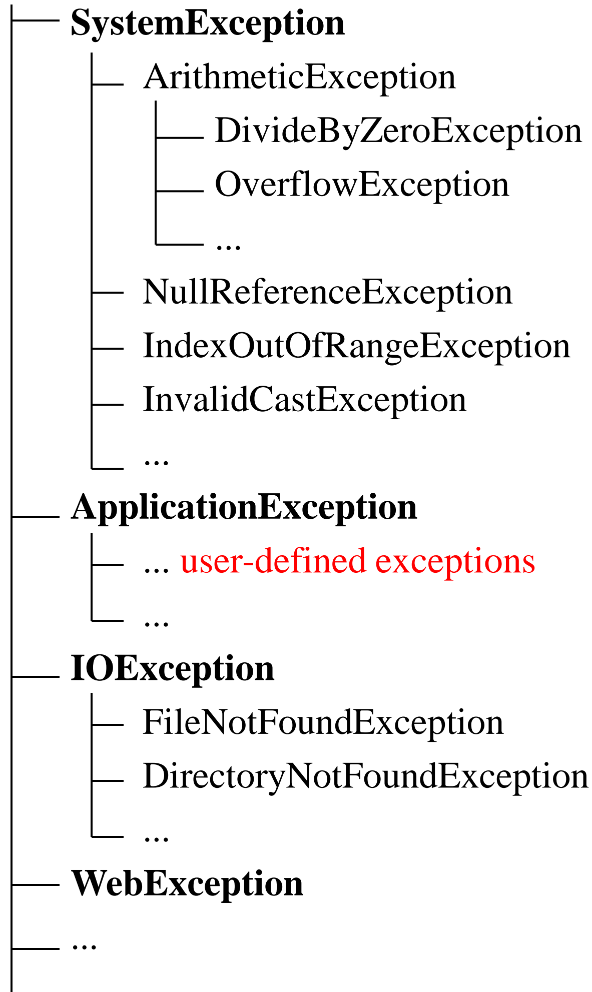
try Statement

```
FileStream s = null;
try {
    s = new FileStream(curName, FileMode.Open);
    ...
} catch (FileNotFoundException e) {
    Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
    Console.WriteLine("some IO exception occurred");
} catch {
    Console.WriteLine("some unknown error occurred");
} finally {
    if (s != null) s.Close();
}
```

- *catch* clauses are checked in sequential order.
- *finally* clause is always executed (if present).
- Exception parameter name can be omitted in a *catch* clause.
- Exception type must be derived from *System.Exception*.
  If exception parameter is missing, *System.Exception* is assumed.

Department of Distributed and Dependable Systems
D3S

Exception Hierarchy (excerpt)



Department of Distributed and Dependable Systems D3S

Throwing an Exception

By an invalid operation (implicit exception)

- Division by 0
- Index overflow
- Acess via a null reference
- ...

By a throw statement (explicit exception)

```
throw new FunnyException(10);

class FunnyException : ApplicationException {
    public int errorCode;
    public FunnyException(int x) { errorCode = x; }
}
```

(By calling a method that throws an exception but does not catch it)

```
s = new FileStream(...);
```

System.Exception

Properties

e.Message	the error message as a string; set by *new Exception(msg);*
e.StackTrace	trace of the method call stack as a string
e.Source	the application or object that threw the exception
e.TargetSite	the method object that threw the exception
...	
e.InnerException	should be always set if rethrowing another exception

Methods

e.ToString()	returns the name of the exception and the StackTrace
...	

What is the output the following program?

```
class T {
    public int a;

    public int m() {
        try {
            a += 10;
            return a;
        } finally {
            a = 15;
        }
    }
}
```

```
class Program {
    static void Main(string[] args) {
        T t = new T();
        t.a = 10;

        Console.WriteLine(t.a);
        Console.WriteLine(t.m());
        Console.WriteLine(t.a);
    }
}
```

Option	Result
A	10 20 20
B	10 20 15
C	10 15 15
D	*It will not compile.*
E	*It will generate a runtime error (unhandled exception)*

Department of Distributed and Dependable Systems D3S

What is the output the following program?

```
class T {
    public int a;

    public int m() {
        try {
            a += 10;
            return a;
        } finally {
            a = 15;
        }
    }
}
```

```
class Program {
    static void Main(string[] args) {
        T t = new T();
        t.a = 10;

        Console.WriteLine(t.a);
        Console.WriteLine(t.m());
        Console.WriteLine(t.a);
    }
}
```

Option	Result
A	10 20 20
B	10 20 15
C	10 15 15
D	*It will not compile.*
E	*It will generate a runtime error (unhandled exception)*

Department of Distributed and Dependable Systems D3S

What is the output the following program?

```
class Program {
    static bool Test() {
          try {
                   return true;
          } finally {
                   return false;
          }
    }

    static void Main(string[] args) {
          Console.WriteLine(Test());
    }
}
```

Option	Result
A	True
B	False
C	*It will generate a runtime error (unhandled exception).*
D	*It will not compile.*

Department of Distributed and Dependable Systems D3S

What is the output the following program?

```
class Program {
    static bool Test() {
           try {
                     return true;
           } finally {
                     return false;
           }
    }

    static void Main(string[] args) {
           Console.WriteLine(Test());
    }
}
```

Option	Result
A	True
B	False
C	*It will generate a runtime error (unhandled exception).*
D	*It will not compile (“**Control cannot leave the body of a finally clause**”).*

What is the output the following program?

```
class Program {
    static void Run() {
          try {
                Run();
          } finally {
                Run();
          }
    }
    static void Main(string[] args) {
          Run();
          Console.WriteLine("End");
    }
}
```

Option	Result
A	End
B	*Nothing.*
C	*It will not compile.*
D	*It will generate a runtime error (unhandled exception).*
E	*It will generate* another *runtime error.*

Department of Distributed and Dependable Systems D3S

What is the output the following program?

```
class Program {
    static void Run() {
        try {
            Run();
        } finally {
            Run();
        }
    }
    static void Main(string[] args) {
        Run();
        Console.WriteLine(“End”);
    }
}
```

Option	Result
A	End
B	*Nothing (will loop “almost” forever).* ***Up to .NET 1.1***
C	*It will not compile.*
D	*It will generate a runtime error (unhandled exception).* ***Up to .NET 1.1***
E	*It will generate* another *runtime error.* ***Since .NET 2.0 (including)***

Department of Distributed and Dependable Systems
D3S

StackOverflowException

```
class Program {
    static void Main(string[] args) {
        try {
            unboundRecursion();
        } catch (Exception ex) {
            Console.WriteLine("Exception caught: {0}", ex.Message);
        }
    }

    static void unboundRecursion() {
        unboundRecursion();
    }
}
```

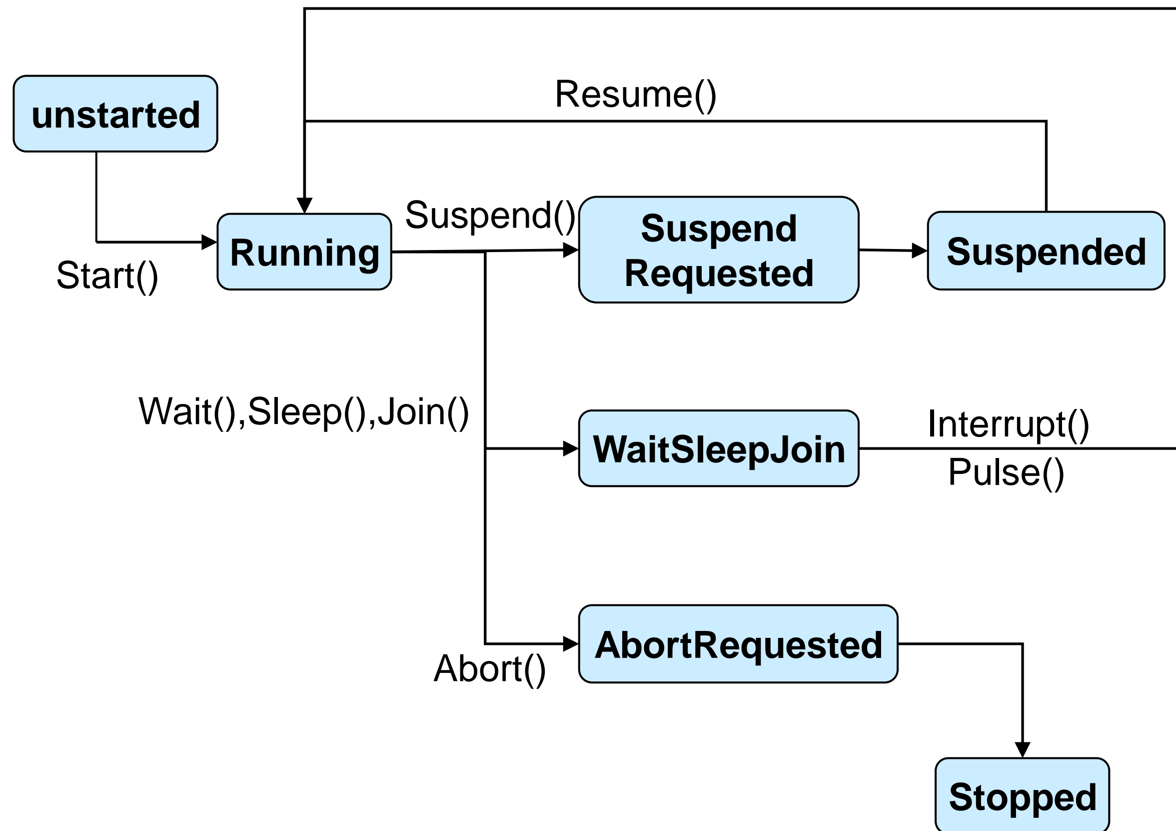
C:\Windows\system32\cmd.exe

```
Process is terminated due to StackOverflowException.
Press any key to continue . . .
```

Thread States

- Enumeration ThreadState defines the states of a thread

```
public enum ThreadState {
        AbortRequested,
        Background,,
        Running,
        Stopped,
        StopRequested,
        Suspended,
        SuspendRequested,
        Unstarted,
        WaitSleepJoin
}
```



Foreground and Background Threads

- Two different types of threads: *Foreground* und *Background*
  - As long as a foreground thread is running, the program will not terminate
  - running background threads cannot prevent the program from terminating
- A background thread is created by setting the property IsBackground

```
Thread bgThread = new Thread(new ThreadStart(…));
bgThread.IsBackground = true;
```

Department of Distributed and Dependable Systems

Class Monitor

- Class Monitor realizes basic mechanism for synchronisation

```
public sealed class Monitor {
   public static void Enter(object obj);
   public static bool TryEnter(object obj);

   public static void Exit(object obj);

   public static void Wait(object obj);
   public static bool Pulse(object obj);
   public static void PulseAll(object obj);
}
```

- tries to get lock for obj and blocks
- tries to get lock for obj and returns
- releases lock for obj
- brings thread into the waiting state, releases locks
- awakens next thread waiting for obj
- awakens all threads waiting for obj

- lock statement is realized using Monitor; is short form for:

```
lock (obj) {
      …
}
```

```
Monitor.Enter(obj)
try {
      …
} finally {
      Monitor.Exit(obj)
}
```