Java for C# programmers

and vice versa…

Acknowledgement: some content taken from
http://www.javacamp.org/javavscsharp

0: General considerations

History

A great many Java developers considered Microsoft's Visual J++ to be the most productive Java IDE (integrated development environment) on the market.

The settlement of a lawsuit (2001) brought by Sun Microsystems stopped Visual J++'s evolution in its tracks and appeared to end Microsoft's involvement with Java.

Microsoft then produced J#, and more or less at the same time C#...

Curiosity: Due to technical limitations of display (standard fonts, browsers, etc.) and the fact that the sharp symbol (U+266F ♯ MUSIC SHARP SIGN (HTML ♯)) is not present on most keyboard layouts, the number sign (U+0023 # NUMBER SIGN (HTML #)) was chosen to approximate the sharp symbol in the written name of the programming language.

Proprietary language?

C# is defined by ECMA and ISO standards, whereas Java is proprietary, though largely controlled through an open community process.

The C# API is completely controlled by Microsoft, whereas the Java API is managed through an open community process.

Structure

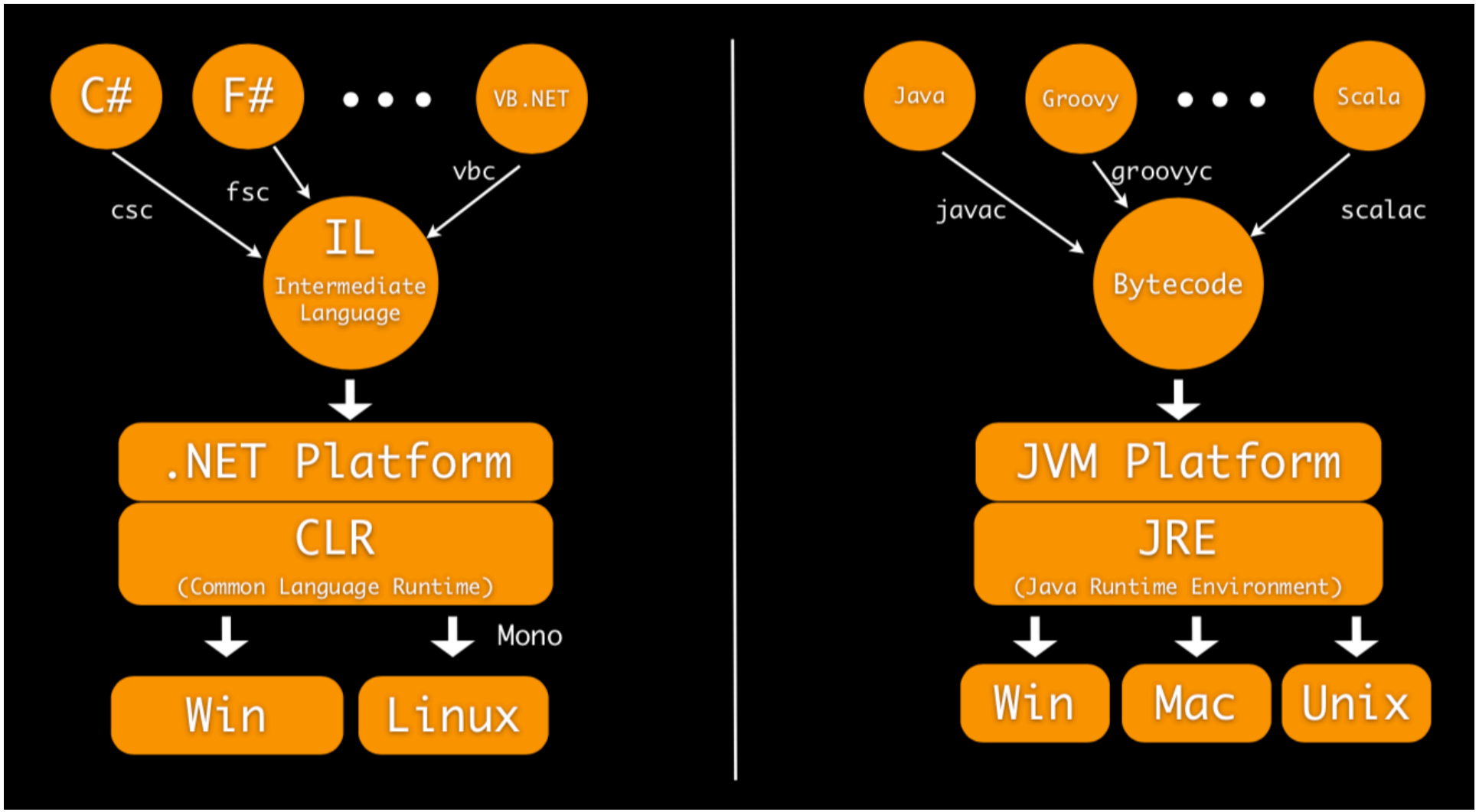


Image by dhaval.dalal@software-artisan.com

Technology stack

	Java	C#
Data Access	JDBC	ADO.NET
Client Side GUI	AWT/Swing	WinForms, WPF
Web Side GUI	JSP, JavaFX, JSF	ASP.NET, WebForms
Web Scripting	Servlets, Filters	ISAPI, HttpHandler, HttpModule
Web Hosting	Tomcat, Jetty, Weblogic etc...	IIS
Remote Invocation	RMI, Netty, AKKA	Remoting, now part of WCF
Messaging	JMS, AKKA	MSMQ
Native	JNI	PInvoke
Directory Access	JNDI	Active Directory (Ldap/ADSI)
Componentization	EJB (Entity/Session), Spring	COM+, Managed Extensibility Framework (MEF)

Image by dhaval.dalal@software-artisan.com

JDK - JRE

JRE: Java Runtime Environment. It is basically the Java Virtual Machine where your Java programs run on. It also includes browser plugins for Applet execution.

JDK: It's the full featured Software Development Kit for Java, including JRE, and the compilers and tools (like JavaDoc, and Java Debugger) to create and compile programs.

Usually, when you only care about running Java programs on your browser or computer you will only install JRE. It's all you need. On the other hand, if you are planning to do some Java programming, you will also need JDK.

JDK-JRE

"The JRE" is basically a bunch of directories with Java-related files:

/bin with executable programs like java and (for Windows) javaw, which are essentially the program that is the Java virtual machine;

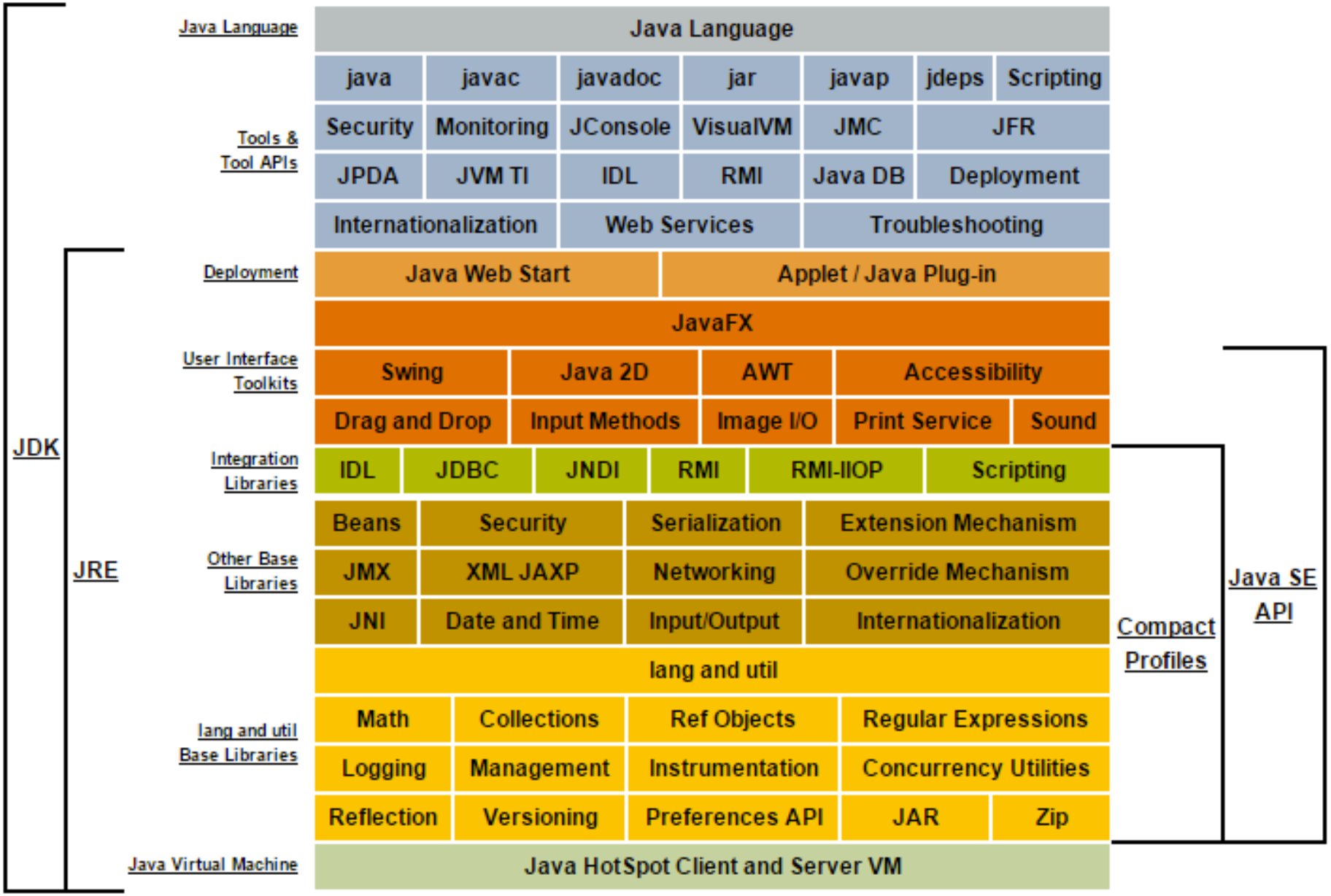
/lib with a large number of supporting files: Some jars, configuration files, property files, fonts, sounds, icons... all the "trimmings" of Java. Most important are rt.jar and a possibly a few of its siblings, which contain the "java API," i.e. the Java library code.

Somewhere are hidden some .DLLs (for Windows) or .so's (Unix/Linux) with supporting, often system-specific native binary code.

The JDK is also a set of directories. It looks a lot like the JRE but it contains a directory (called JRE) with a complete JRE, and it has a number of development tools, most importantly the Java compiler javac in its bin directory.

Description of Java Conceptual Diagram

JDK - JRE



1: Contact points

Wow, it's the same thing!

The languages share many features:

- Object orientation
- Same ancestors (C, C++) => same grammar
  - Some people say C# is derived from Java
- Heap-based classes (but C# also supports stack-based classes, called value types)
- Garbage collection
- String class (strings are immutable)
- No global methods and variables
- Initialization of instance and static variables.
- Superclass (base class) constructor call, and constructor chaining (call to a constructor from inside another constructor)
- Value types for primitive data types, reference types for objects.

2: Trivial differences: syntax only

trivial but annoying differences…

main

Java	C#
public static void main(String[] args) {…}	static void Main(string[] args) {…}
	static void Main() {…}
	static int Main(string[] args) {…}
	static int Main() {…}

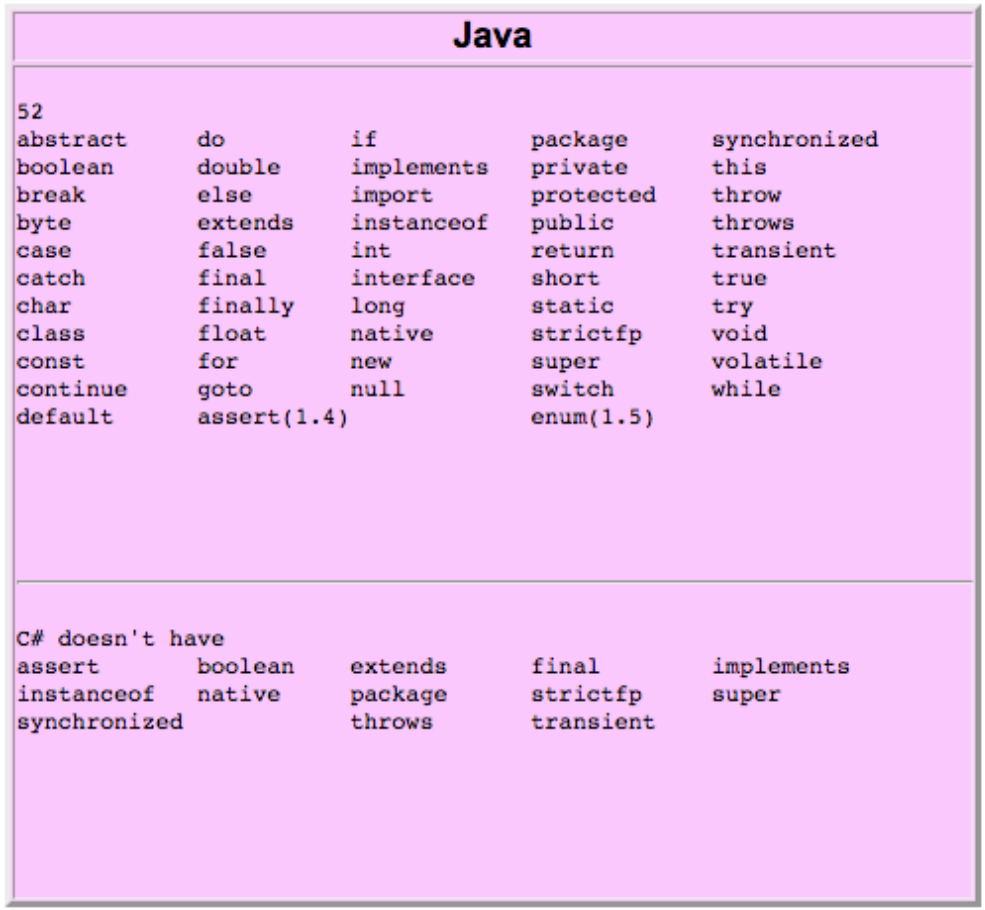
Note: as a convention:
method names are lowercase in Java and uppercase in C#
class names are uppercase in Java and lowercase in C#

Print statements

Java	C#
System.out.println("Hello world!");	System.Console.WriteLine("Hello world!");
	Console.WriteLine("Hello again!");

Note: as a convention:
method names are lowercase in Java and uppercase in C#
class names are uppercase in Java and lowercase in C#

keywords



Java

52

abstract	do	if	package	synchronized
boolean	double	implements	private	this
break	else	import	protected	throw
byte	extends	instanceof	public	throws
case	false	int	return	transient
catch	final	interface	short	true
char	finally	long	static	try
class	float	native	strictfp	void
const	for	new	super	volatile
continue	goto	null	switch	while
default	assert(1.4)		enum(1.5)	

C# doesn't have

assert	boolean	extends	final	implements
instanceof	native	package	strictfp	super
synchronized		throws	transient	

strictfp

C#

77

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Java doesn't have

as	base	bool	checked	decimal
delegate		event	explicit	extern
fixed	foreach	implicit	int	internal
is	lock	namespace	object	operator
out	override	params	readonly	ref
sbyte	sealed	sizeof	stackalloc	string
struct	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	

goto

switch

Java	C#

switch (i) {
case 1:
case 2:
a=1;
case 3:
b=2;
break;
case 4-7:
b=9;
break;
default:
c=3;
}
	<= fallback is not allowed for non-empty cases (break is missing)

Switch statements with String cases have been implemented since Java SE 7

Array declarations

Java	C#
int[] iArray = new int[100]; float fArray[] = new float[100];	int[] iArray = new int[100];
int[][][] a3 = new int[20][20][30];	int[][][] a3 = new int[20][20][30];
	int[,] myRectangularArray = new int[rows, columns];

Multidimensional arrays in Java and C# are jagged

```
int[][]j2 = new int[3][];
    j2[0] = new int[] {1, 2, 3};
    j2[1] = new int[] {1, 2, 3, 4, 5, 6};
    j2[2]= new int[10];
```

C# also has rectangular bidimensional arrays (more efficient)

for - foreach

Java	C#

int [] k={1,2,3};
for (int z:k){
System.out.println(z);
}
	int [] k={1,2,3};
	foreach (int z in k){
	System.Console.WriteLine(z);
	}

for (object o:collection){…}	foreach (object o in collection){…}

Constants

Java	C#
static final double PI=3.1415926535;	const double PI=3.1415926535;
final int N=2;	readonly int Z=24;

Java:
Constants with values which may be different for different instances are not static.

C#:
When the constant needs to be initialized at run-time use the readonly keyword instead of const.

final/readonly vars can only be set at declaration time or in the constructor

Packages-namespaces

Java	C#
package	namespace
import	using

The C# package structure is defined using namespaces (just like Java), but the namespaces do NOT have to reflect the directory structure.

C#'s namespaces are more similar to those in C++. Unlike Java, the namespace does not specify the location of the source file. (Actually, it's not strictly necessary for a Java source file location to mirror its package directory structure.)

C# has the ability to alias namespaces.

Packages-namespaces nesting

Only in C#: namespace syntax also allows to nest namespaces

```
using System;
  namespace One{
    public class MyClass {…} /*One.MyClass */
    namespace Two{
       public class MyOtherClass {…} /*One.Two. MyOtherClass */
       } // end of namespace Two
   } // end of namespace One
```

Assemblies

Java	C#
Assemblies are stored as JAR archives (Java ARchive: zip format with a manifest). A JAR can contain a library, or can be an “executable” file. Some JAR are specialized for deployment in containers: WAR (Web ARchive) EJB-JAR (Entity Beans Archives) EAR (Enterprise ARchive)	Assemblies are usually stored as EXEs or DLLs

Boxing-Unboxing

Java	C#
int i = 123; Object o=i; // Object o=123;	int i = 123; object o = i; // object o=123;

Object o = 123;
int i = (Integer)o;
 	
// but also
Integer o = 123;
int i = o;
	object o = 123;
	int i = (int)o;

In C# unboxing is always explicit

varargs

Java	C#
void f(String title, Integer…args)	void f(string title, params int[] args)

Invoking Garbage Collector

Java	C#
System.gc()	GC.Collect()

Concurrency - synchronization

Java	C#
synchronized (this) { ... }	lock (this) { ... }
synchronized void method() { ... }	[MethodImpl(MethodImplOptions.Synchronized)] void Method() { ... }

Serialization

Java	C#

MyClass implements Serializable {
transient Object x;
…
}
	[Serializable]
	myClass {
	[ScriptIgnore]
	object x;
	…
	}

[Serializable]: is used to mark a class as serializable and is similar to a Java class implementing the Serializable interface.

Put [ScriptIgnore] or [NonSerialized] attribute on the property and it will not be serialized. (transient in Java)

Class loading

Java	C#
Class x=Class.forName(“class_name”); x.getInstance(); ClassLoader.getResources()	Activator.CreateInstance(Type t) Assembly.Load()

Run-time type identification

Java	C#
if(x instanceof MyClass) MyClass mc = (MyClass) x;	if(x is MyClass) MyClass mc = (MyClass) x;

Metadata

Java	C#

Class
Class klass = X.class; // call on class
Class klass= x.getClass(); //call on instance
	Type
	Type type = typeof(X); // call on class
	Type type = i.GetType() ; //call on instance

Metainformation

Java	C#
@Annotation	[Attribute]

3: Java features absent in C#

There is no such a thing as…

Cross Platform Portability:
Write Once, Run Anywhere

Nested classes

Class declaration inside another class.

Java inner class:

for each instance of the enclosing class there exists a corresponding instance of the inner class that has access to the enclosing class's instance variables and contains no static methods.

Java static nested class:

the nested class has access to the static members and methods of the enclosing class.

C# ONLY has static nested classes.

Anonymous inner classes

```
btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });
```

4: Simple but relevant differences

Really important differences…

Primitive data types

- Not every C# primitive type has a corresponding type in Java: Java does not have the unsigned types (ulong, uint, ushort and byte).
- The corresponding ones hava the same name (except for byte: the byte type in Java is signed and is thus analagous to the sbyte type in C# and not the byte type).
- C# also has is the decimal type, a type which stores decimal numbers without rounding errors (at the cost of more space and less speed).
- NOTE: primitive data types in C# are subclasses of object, in Java they are not!

Primitive data types

- NOTE: primitive data types in C# are subclasses of object, in Java they are not!
- This means that an int in C# inherits all the methods of object.
- It's important to realize, however, that the primitive types are still value types. Boxing (the conversion of a value type to a reference type) occurs in both C# and Java.

Big numbers

Both Java and C# have BigInteger.

It’s for precision (large) integers.

Java also has, but C# hasn’t, BigDecimal.

Access modifier

	Java	C#
public	Access not limited	Access not limited
protected	Access limited to the package and to subclasses also in a different package	Access limited to the containing class and its subclasses
protected internal		Access limited to this program or types derived from the containing class
internal		Access limited to this program (assembly)
(no access modifier)	Access limited to the package	Same as private
private	Access limited to the containing type	Access limited to the containing type

Friend Assembly (C# only)

The friend assembly feature allows an internal type or internal member of an assembly to be accessed from another assembly.

To give one assembly access to another assembly's internal types and members, the [InternalsVisibleToAttribute] attribute is used.

Inheritance

Java	C#
class B extends A implements Comparable {…}	class B:A, IComparable

Note:

- different syntax for Interface inheritance and class inheritance in Java
- Usually interfaces names start with I in C#
- For both Java and C# inheritance is
  - SINGLE for classes
  - MULTIPLE for Interfaces

Java	C#
Everything derives from Object. This includes classes you create, not including primitive types.	Everything ultimately derives from Object. This includes classes you create, as well as value types such as int or structs.

Partial classes: a C# only feature

- With C# you can define a class across multiple files.

- The real use of this is when you are generating code and you want to be able to have part of a class generated, but still have the other part of the class manually controlled and versioned.

Unextendable classes

Java	C#
final class B {…}	sealed class B {…}

Static constructor

Java	C#

Class StaticInitTest {
static{ // called before first use of static class
System.out.println("In static constructor");
}
…}
	Class StaticInitTest {
	static StaticInitTest(){
	Console.WriteLine("In static constructor");
	}
	…}

Constructor chaining

Java	C#
super()	base()

```
public class Child extends Parent
{
   public Child() {
      super("param");
      System.out.println("Child Constructor.");
   }
  … super.f();…}
```

```
public class Child : Parent
{
   public Child() : base(”param")
   {
      Console.WriteLine("Child Constructor.");
   }
  … base.f();…}
```

Destructor
Non-Deterministic Object Cleanup

Java	C#
finalize()	~X()

In C#, destructors automatically call the base class destructor after executing, which is not the case in Java.

Constraint on public classes

In Java, there can only be one class per source file that has public access and it must have the same name as the source file.

C# does not have such constraint.

Virtual methods

Java	C#

All methods are by default virtual and you can override them (dynamic binding).
 	
Java has static binding for:
final, static and private methods.
 	
You can annotate overriding methods:
@Override
public void myMethod() { ... }
	All methods are non-virtual (static binding).
	
	To override a method in the parent class, make sure the method of the parent class is defined as virtual using the virtual keyword.
	
	class MyParentClass {
	public virtual void MyMethod() { ... }
	}
	In the child class, the method must use the override keyword.
	
	class MyChildClass : MyParentClass {
	public override void MyMethod() { ... }
	}

Enumerations

Java	C#
enum, can have fields, methods and implement interfaces and are typesafe	enum, cannot have methods, fields or implement interfaces, not typesafe

- Java made enumerations just like classes, except they do not have inheritance.
- C# enumerations are more along the lines of C/C++ implementations in which they are just basically integers, but they support the ToString method, so they can report their value as string (such as “North") and not just an an integer.

Equality check

Java	C#
Value equality: b == a Reference equality (identity): b==a; Equality for reference types: override Object.equals(Object) override Object.hashcode()	Value equality: b == a Reference equality (identity): System.Object.ReferenceEquals(a, b); Equality for reference types: override Object.Equals(Object) override Object.GetHashCode() Optionally overload the == and != operators

Generics

- Although, the Generics feature in both C# and Java is similar in concept to templates in C++.
- Generics in Java and C# looke very similar, but implementations are however quite different.

Java	C#
the generic functionality is implemented using type erasure. The generic type information is present at compile time, after which it is erased and all the type declarations are replaced with Object. Casts are automatically inserted in the right places. Hence, runtime introspection does not reveal the generic.	permits full runtime introspection of generic types and generic type parameters (useful e.g. for instance creation and array creation).

- C# implementation is also more efficient (doesn't do any cast).

Weak references

Java	C#
someCollection.add(inew WeakReference(new MyData(i)); Java also has: • SoftReference • PhantomReference	someCollection.Add(new WeakReference(new MyData(i));

For an explanation see also:
https://web.archive.org/web/20061130103858/http://weblogs.java.net/blog/enicholas/archive/2006/05/understanding_w.html

Different operators: >>> and ->

Java only:

The >>> operator is the unsigned right bit-shift. It effectively divides the operand by 2 to the power of the right operand.

The difference between >> and >>> would only show up when shifting negative numbers. The >> operator shifts a 1 bit into the most significant bit if it was a 1, and the >>> shifts in a 0 regardless.

C# only:

The -> operator combines pointer dereferencing and member access (as in C, C++).

The -> operator can be used only in code that is marked as unsafe.

The -> operator cannot be overloaded.

Checked-unchecked exceptions

The checked exceptions that a method may raise are part of the method's signature. For instance, if a method might throw an IOException, it must declare this fact explicitly in its method signature. Failure to do so raises a compile-time error.

C# does not include checked exceptions, Java does.

throw/throws

```
throw statement
class Test
{
    static void F() throws Exception {
        try {
            G();
        }
        catch (Exception e) {
            System.out.println("Exception in F: " + e.getMessage());
            e = new Exception("F");
            throw e;   // re-throw
        }
    }
    static void G() throws Exception{
        throw new Exception("G");
    }
    public static void main(String[] args) {
        try {
            F();
        }
        catch (Exception e) {
            System.out.println("Exception in Main: " + e.getMessage());
        }
    }
}
>java Test
Exception in F: G
Exception in Main: F
```

```
throw statement
class Test
{
    static void F() {
        try {
            G();
        }
        catch (Exception e) {
            Console.WriteLine("Exception in F: " + e.Message);
            e = new Exception("F");
            throw;      // re-throw
        }
    }
    static void G() {
        throw new Exception("G");
    }
    static void Main() {
        try {
            F();
        }
        catch (Exception e) {
            Console.WriteLine("Exception in Main: " + e.Message);
        }
    }
}
>csc Test.cs
Exception in F: G
Exception in Main: F
```

Nullable types (C#)

A nullable types is an instance of the System.Nullable type.

A nullable type can represent the normal range of values for an underlying value type as well as the null value.

For example, the type Nullable<bool> can represent the values true, false and null.

A nullable type can be declared by appending the operator '?' to the name of a value type when declaring the variable.

bool? is equivalent to Nullable<bool>.

null coalescing operator: ?? (C#)

```
string pageTitle = suppliedTitle ?? "Default Title";

// is equivalent to

string pageTitle = (suppliedTitle != null) ? suppliedTitle :
"Default Title";

// and is equivalent to

string pageTitle;
if (suppliedTitle != null) pageTitle = suppliedTitle;
else  pageTitle = "Default Title”;
```

Same in Java (Optional)

```
String myTitle=null;
Optional<String> suppliedTitle=Optional.ofNullable(myTitle);
String title = suppliedTitle.orElse("Default Title");
System.out.println(title);

// also:
String title2 = suppliedTitle.orElseGet(()-> {
        return new String("Default Title2");});
System.out.println(title2);
```

Optional has several other methods

Documentation generation

Java	C#
** * Calculates the square of a number. * @param num the number to calculate. * @return the square of the number. * @exception NumberTooBigException this occurs if the square of the number * is too big to be stored in an int. */ public static int square(int num) throws NumberTooBigException{}	///<summary>Calculates the square of a number.</summary> ///<param name="num">The number to calculate.</param> ///<return>The square of the number. </return> ///<exception>NumberTooBigException - this occurs if the square of the number ///is too big to be stored in an int. </exception> public static int square(int num){}

Javadoc vs C# XML

C# XML is more limited than Javadoc

Documentation generation

Javadoc allows one to document the following metadata about a method:

- Description of the method.
- Exceptions thrown by the method.
- Parameters the method accepts
- Return type of the method.
- Associated methods and members.
- Indication as to whether the API has been deprecated or not.
- Version of the API the method was first added.
- The deprecated information is also used by the compiler which issues a warning if a call to a method marked with the deprecated tag is encountered during compilation.

Javadoc also provides the following information automatically:

- Inherited API
- List of derived classes
- List of implementing classes for interfaces
- Serialized form of the class
- Alphabetical class listing.
- Package hierarchy in a tree format.

5: C# features absent in Java

There is no such a thing as…

Preprocessor directives

- C# has proprocessor directives (similar to those of C and C++)

Although the compiler doesn't have a separate preprocessor, the directives are processed as if there were one. They are used to help in conditional compilation.

Unlike C and C++ directives, you cannot use these directives to create macros.

Implicitly typed variables

```
var i = 10; // implicitly typed
int i = 10; //explicitly typed
```

Note: in Java type inference is only done with generics, but the other way round:

```
 LinkedList<String> x=new LinkedList<>();
```

Properties (C# only)

Java	C#

public class PropHolder {
private int someProperty = 0;
public int getSomeProperty(){
return someProperty; }
public void setSomeProperty(int x) {
someProperty = x; }
}
 	
public class PropertyTester {
public static void main(String[] args) {
PropHolder propHold = new PropHolder();
propHold.setSomeProperty (5);
System.out.println("Property Value: ",
propHold.getSomeProperty);
}
}
	public class PropHolder {
	private int someProperty = 0;
	public int SomeProperty
	{
	get { return someProperty; }
	set {someProperty = value; }
	}
	}
	public class PropertyTester {
	public static void Main(string[] args) {
	PropHolder propHold = new PropHolder();
	propHold.SomeProperty = 5;
	Console.WriteLine("Property Value: {0}",
	propHold.SomeProperty);
	}
	}

You can ask the IDE to create getters and setters for you

Properties (C# only)

To the client, a property looks like a member variable, but to the implementor of the class it looks like a method.

It allows you total encapsulation and data hiding while giving your clients easy access to the members.

The value is implicitly available to the property.

You can have readonly and writeonly properties.

Verbatim strings

Java	C#
String path = "C:\\My Documents\\"; //There is no verbatim string in Java	string path = "C:\\My Documents\\"; //can be written with verbatim string like: string path = @"C:\My Documents\";

Also, some difference in escape sequences:

Java

Character Escape	Sequence
single quote	\'
double quote	\"
backslash	\\
Backspace	\b
Form feed	\f
New Line	\n
Carriage Return	\r
Horizontal Tab	\t
Unicode	\u
hexadecimal	\x

C#

Character Escape	Sequence
single quote	\'
double quote	\"
backslash	\\
Alert	\a
Backspace	\b
Form feed	\f
New Line	\n
Carriage Return	\r
Horizontal Tab	\t
Vertical Tab	\v
Unicode	\u
hexidecimal	\x
null	\0 (zero)

Out parameters (pass by reference)

C# only: the **out** keyword, which indicates that you may pass in uninitialized variables and they will be passed by reference.

The calling method should mark out keyword.

```
class Test {
   static void Divide(int a, int b, out int result, out int remainder) {
      result = a / b;
      remainder = a % b;
   }
   static void Main() {
      for (int i = 1; i < 10; i++)
         for (int j = 1; j < 10; j++) {
            int ans, r;
            Divide(i, j, out ans, out r);
            Console.WriteLine("{0} / {1} = {2}r{3}", i, j, ans, r);
         }
   }
}
```

Struct

```
struct Point {
    public int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

Point a = new Point(10, 10);
Point b = a;
a.x = 100;
System.Console.WriteLine(b.x);
```

A struct is a user-defined value type.

Structs in C# are used similarly to classes, except that it can't inherit from any class, nor can any class inherit from it (they are “sealed” and the implicit base class is System.ValueType, which is derived from Object.

They can contain constructors, constants, methods and more.

struct is not a reference type!

Aliases

The using keyword can be used to alias the fully qualified name for a type similar to the way typedef is used in C and C++.

This is useful in creating readable code where the fully qualified name of a class is needed to resolve namespace conflicts.

```
using Terminal = System.Console;

class Test{
    public static void Main(string[] args){
      // Terminal.WriteLine is equivalent to System.Console.Writeline
      Terminal.WriteLine(”hello");
    }
}
```

Static classes

A static class is a class that has no instance members, no instance constructors and cannot be used as a base class. (e.g. System.Math class)

Overflow detection

C# provides the option to explicitly detect or ignore overflow conditions in expressions and type conversions.

```
/* overflow detected only if /checked compiler option on */
  byte a = (byte) num;

  checked{
     byte b = (byte) num; /* overflow ALWAYS detected */
  }

  unchecked{
     byte c = (byte) num; /* overflow NEVER detected */
  }
```

Pointers and unsafe code

It is possible to have in C++ and C-like pointer types within an "unsafe context" (a lot of runtime checking is then disabled).

```
public static unsafe void Sort(int* array, int size){
  for(int i= 0; i < size - 1; i++)
    for(int j = i + 1; j <size; j++)
    if(array[i] > array[j] )
      Swap(&array[i], &array[j] );
  }
```

Since garbage collection may relocate managed variables during the execution of a program, the fixed keyword is provided so that the address of a managed variable is pinned during the execution of the parts of the program within the fixed block.

Deterministic Object Cleanup

By implementing the interface iDispose you can expose a method (Dispose) to actually delete an object without waiting for the Garbage Collector to come.

Also, you can tell the GC to keep clear from this object:

GC.SuppressFinalize(this);

See https://stackoverflow.com/questions/538060/proper-use-of-the-idisposable-interface

Explicit interface

A class may implement in different way a method belonging to different interfaces. The implementing method must be private, and can only be called by casting on the interface.

```
public class SampleClass : IControl, Isurface {
   void IControl.Paint() {
      System.Console.WriteLine("IControl.Paint");
   }
   void ISurface.Paint() {
      System.Console.WriteLine("ISurface.Paint");
   }
}
```

```
// Call the Paint methods from Main.
SampleClass obj = new SampleClass();
//obj.Paint();  // Compiler error.
IControl c = (IControl)obj;
// Calls IControl.Paint on SampleClass.
c.Paint();
ISurface s = (ISurface)obj;
// Calls ISurface.Paint on SampleClass.
s.Paint();
```

Output:

```
IControl.Paint
ISurface.Paint
```

Operator overloading

Java does not include operator overloading, because abuse of operator overloading can lead to code that is harder to understand and debug.

(but the language defines the + operator for Strings)

C# allows operator overloading, which, when used carefully, can make code terser and more readable.

Example: + overloading

```
public static Box operator+ (Box b, Box c) {
   Box box = new Box();
   box.length = b.length + c.length;
   box.width = b.width + c.width ;
   return box;
}
```

Some operators cannot be overloaded:

The conditional logical operators: &&, ||
The assignment operators: =,+=, -=, *=, /=, %=
 ., ?:, ->, new, is, sizeof, typeof

Example: == overloading

```
public class Score : Icomparable {
   int value;
   public Score (int score) {
      value = score;
   }
   public static bool operator == (Score x, Score y) {
      return x.value == y.value;
   }
   public static bool operator != (Score x, Score y) {
      return x.value != y.value;
   }
   public int CompareTo (object o) {
      return value - ((Score)o).value;
   }
}

Score a = new Score (5);
Score b = new Score (5);
Object c = a;
System.Console.WriteLine (((IComparable)c).CompareTo (a));   //0
System.Console.WriteLine ((object)a == (object)b;            // false
System.Console.WriteLine (a == b);                           // true
```

Some operators must be redefined in pairs:

!= and ==

> and <

>= and <=

Indexer ([] overloading)

Indexers allow instances of a class or struct to be indexed just like arrays. The indexed value can be set or retrieved without explicitly specifying a type or instance member. Useful e.g. to define array-like stuctures with non-numeric indexes.

```
class DayCollection {
   string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat" };
   private int GetDay(string testDay) {
      for (int j = 0; j < days.Length; j++) {
         if (days[j] == testDay) { return j; }
      }
      throw new System.ArgumentOutOfRangeException(
            testDay,
            ”unecognized string");
   }
   public int this[string day] {
      get { return (GetDay(day)); }
   }
}
```

```
var week = new DayCollection();
System.Console.WriteLine(week["Fri"]);
```

(example from docs.microsoft.com)

Delegates

Delegates are a mechanism for providing callback functions (similar to function pointers in C or functors in C++).

```
// delegate declaration, similar to a function pointer declaration
    public delegate void CallbackFunction(Dog d);

//function compatible with delegate declaration
    public static void Bark(Dog d){
        Console.WriteLine("{0} says WOOF",d);
 }

 //create delegate using delegate inference
   CallbackFunction myCallback = Bark;
   myCallback(fido); //invoke it
```

Anonymous methods

Go hand in hand with delegates:

```
CallbackFunction cf=delegate(Dog d) {
    Console.WriteLine("{0} says WOOF",d);
}
cf(fido);
```

Lambda expressions (also in Java)

```
delegate void X();
X instanceOfX;
instanceOfX = delegate() { code };

instanceOfX = () => { code }; // alternative, using lambda expression
```

Lambda expressions are also available in Java

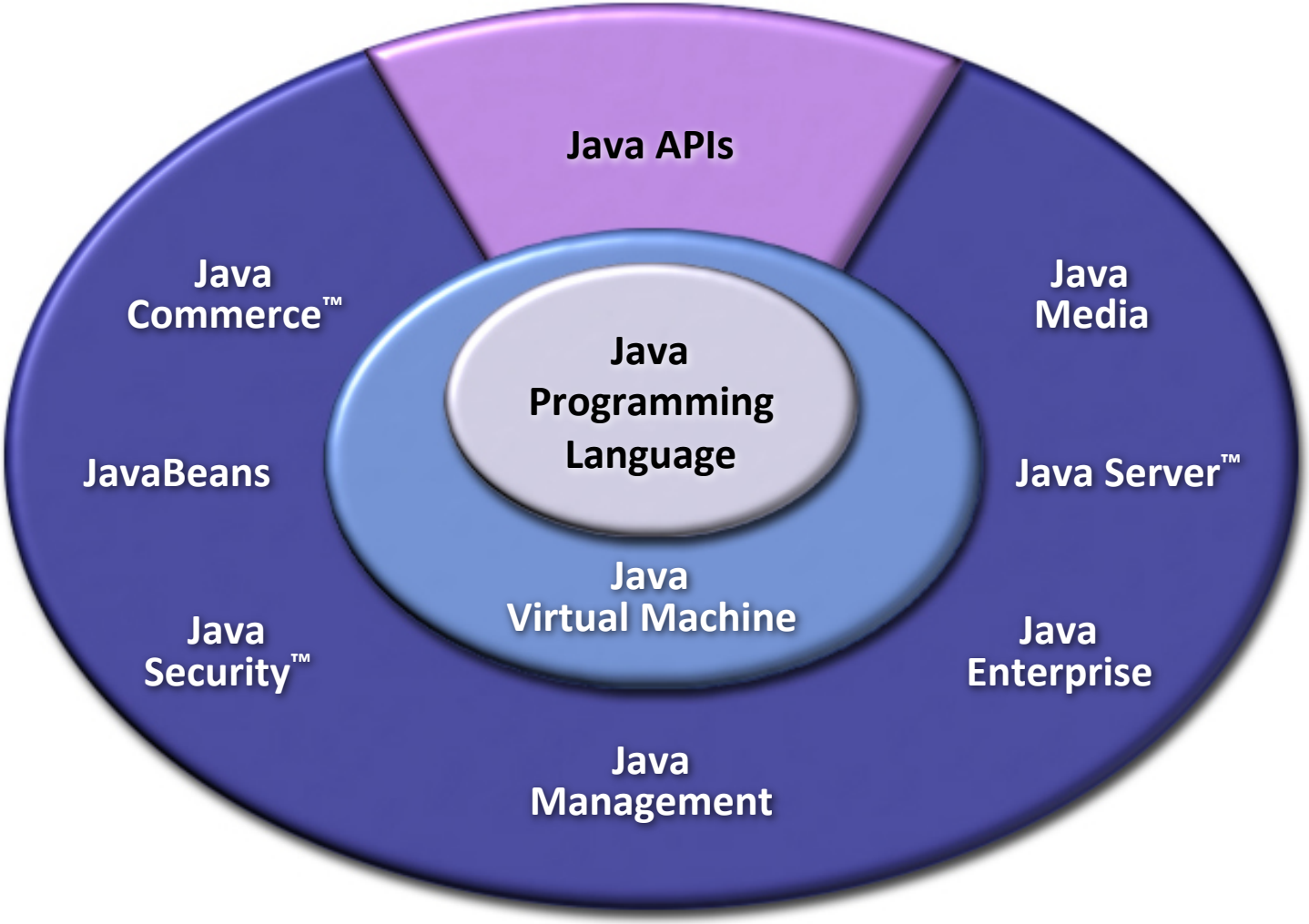
```
() -> { code };
```

See also
https://blogs.msdn.microsoft.com/ericlippert/2007/01/10/lambda-expressions-vs-anonymous-methods-part-one/

6: And the, of course, the API…

This is big!

APIs (Java)


Java APIs
Java Commerce™
Java Media
Java Programming Language
JavaBeans
Java Server™
Java Virtual Machine
Java Security™
Java Enterprise
Java Management

e.g.: collections

Java	C#
The Java collections framework consists of a large number of the classes and interfaces in the java.util package. Its classes have been retrofitted to support generics. The Java collections framework has a number of algorithms for manipulating the elements within its collections, such as: performs sorts and binary searches - find the largest/smallest element based on some Comparator, find sublists within a list, reverse the contents of a list, shuffle the contents of a list, create immutable versions of a collection.	The Systems.Collections namespace contains: • interfaces and abstract classes that represent abstract data types such as IList, IEnumerable, IDictionary, ICollection, and CollectionBase • concrete implementations of data structures such as ArrayList, Stack, Queue, HashTable and SortedList. The System.Collections.Generic namespace contains generic implementations of the above.