Introduction to C#

Basics

- C# was released by the Microsoft in the middle of 2000.
- Architect of c# was Anders Hejlsberg
- C# is an elegant and type-safe object-oriented language that enables developers to build a wide range of secure and robust applications that run on the .NET Framework.
- C# is used to create traditional Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much more
- C# is also simple and easy to learn.
- The coding is similar to Java.
- As an object-oriented language, C# supports the concepts of encapsulation, inheritance and polymorphism.
- All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions
- It has no global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions

- ***Garbage collection*** automatically reclaims memory occupied by unused objects;
- ***Exception handling*** provides a structured and extensible approach to error detection and recovery;
- ***Type-safe*** design of the language makes it impossible to read from uninitialized variables, to index arrays beyond their bounds, or to perform unchecked type casts.
- C# has a ***unified type system***. All C# types, including primitive types such as int and double, inherit from a single root object type.
- Thus, all types share a set of common operations, and values of any type can be stored, transported, and operated upon in a consistent manner.
- C# supports both user-defined reference types and value types, allowing dynamic allocation of objects as well as in-line storage of lightweight structures.

Versions

- C# 1.0, C# 1.1 , C# 2.0, C# 3.0 , C# 4.0
- C# 5.0, C# 6.0
- C# 7.0, C# 7.1, C# 7.2
- C# 7.3   - .NET Framework 4.7.2
- C# 8.0   - .NET Core 3.0
- C# 9.0   - .NET 5.0
- **C# 10.0   .NET 6.0   Visual Studio 2022**

C# versus Java (Similarity)

- C# and Java are both languages descended from C and C++.
- Each includes advanced features, like garbage collection, which remove some of the low level maintenance tasks from the programmer. In a lot of areas they are syntactically similar.
- Both C# and Java compile initially to an intermediate language:
  - C# to Intermediate Language (IL), and Java to Java bytecode.
  - In each case the intermediate language can be run - by interpretation or just-in-time compilation - on an appropriate virtual machine. In C#, however, more support is given for the further compilation of the intermediate language code into native code.
- Like Java, C# gives up on multiple class inheritance in favor of a single inheritance model. C# supports the multiple inheritance of interfaces.

C# versus Java (Differences)

- C# contains more primitive data types than Java, and also allows more extension to the value types.
    - For example, C# supports enumerations, type-safe value types which are limited to a defined set of constant variables, and structs, which are user-defined value types .
    - Java doesn't have enumerations, but can specify a class to emulate them .
- Unlike Java, C# has the useful feature that we can overload various operators.
- However, polymorphism is handled in a more complicated fashion, with derived class methods either **overriding** or **hiding** super class methods.
- In Java, multi-dimensional arrays are implemented solely with single-dimensional arrays where arrays can be members of other arrays. In addition to **jagged arrays**, however, C# also implements genuine rectangular arrays.

Program

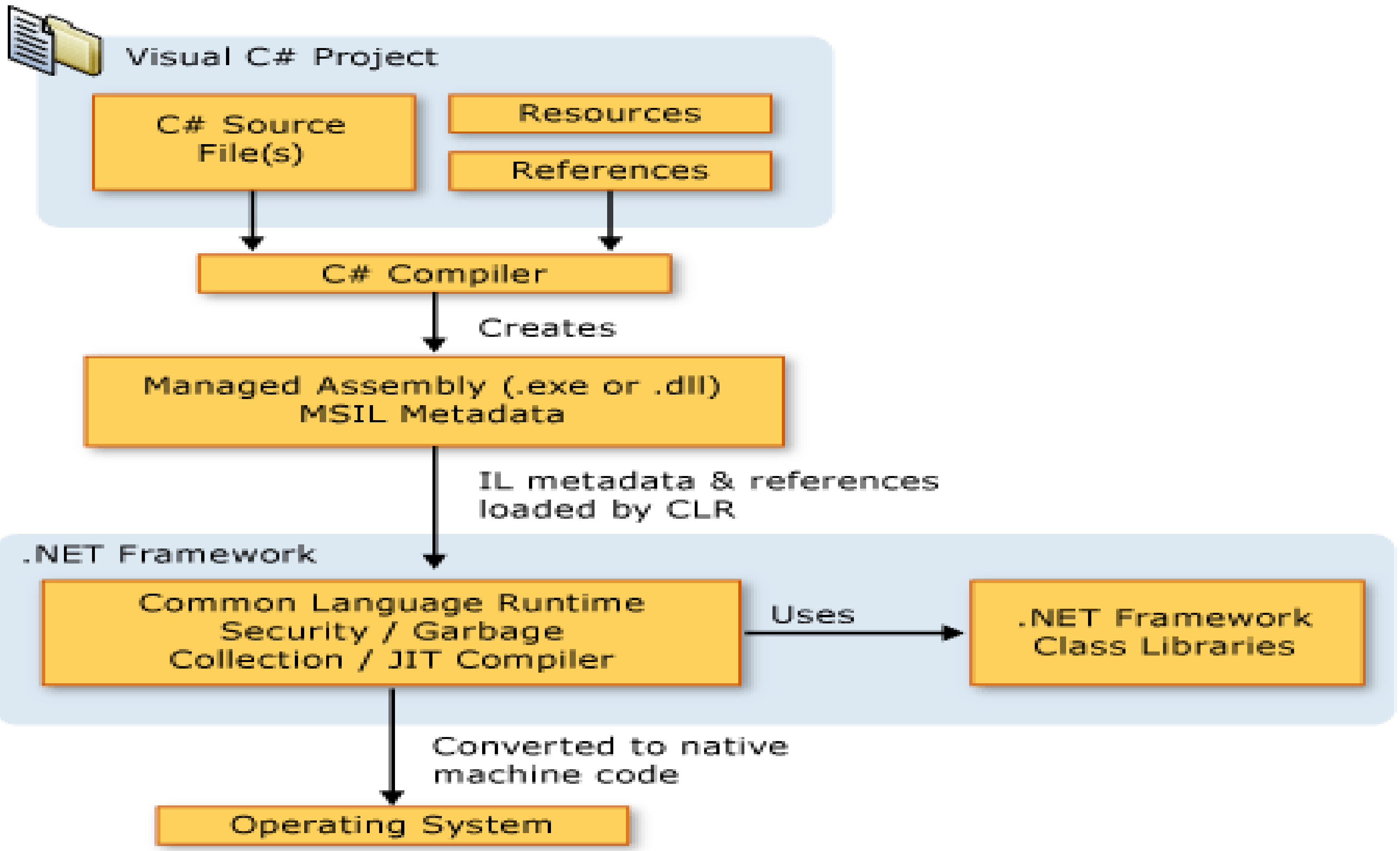
```
using system;
class Hello
{
    static void Main()
    {
        Console. WriteLine("Hello, World");
}
}
```

- First line is using directive that references the System namespace
- Second is the class name
- Fourth- The Main method is declared with the static modifier.
- While instance methods can reference a particular enclosing object instance using the keyword this, static methods operate without reference to a particular object.
- static method named Main serves as the entry point of a program.
- Writeline is the method of the Console Class.
- C# supports two types of Comments:

    Delimited - /* Single line or multiple lines   */

    Single line - //

- The **Main** method is the entry point of your program, where the program control starts and ends.
- It is declared inside a class or struct. The enclosing class or struct is not required to be static.
- It must be static.
- It can either have a void or int return type.
- The **Main** method can be declared with or without parameters.
- Parameters can be read as zero-indexed command line arguments.
- Unlike C and C++, the name of the program is not treated as the first command line argument.
- Libraries and services do not require a Main method as an entry point.
- There can only be one entry point in a C# program. If you have more than one class that has a Main method, we have to compile the program with the **/main** option to specify which Main method to use as the entry point


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

- static void Main()
- static int Main()
- **static void Main(string[] args)**
- static int Main(**string[] args**)

Formatted Output

- Console.WriteLine(“Here is 10/3:{0:#.##}”,10.0/3.0);
- Console.WriteLine(“Current Balance is {0:C}”, balance);
- Console.Write("{0:F2}", 25);   output : 25.00
- Console.Write("{0:D5}", 25);   output : 00025
- Console.Write("{0:F0}", 25);   output : 25

Command Line arguments Rules

- Arguments are delimited by white space, which is either a space or a tab.
- The caret character (^) is not recognized as an escape character or delimiter.
- A string enclosed in double quotation marks ("string") is interpreted as a single argument, regardless of white space that is contained within. A quoted string can be embedded in an argument.
- A double quotation mark preceded by a backslash (\") is interpreted as a literal double quotation mark character (").
- Backslashes are interpreted literally, unless they immediately precede a double quotation mark.

Keywords

- Keywords are predefined, reserved identifiers that have special meanings to the compiler.
- They cannot be used as identifiers in your program unless they include @ as a prefix. For example, @if is a valid identifier but if is not because if is a keyword.

abstract as bool float break continue private protected

internal public sealed readonly ref sizeof voltaile using

overide virtual internal namespace void

Structure of C# Programming

- C# programs can consist of one or more files.
- Each file can contain zero or more namespaces.
- A namespace can contain types such as classes, structs, interfaces, enumerations

General structure:

```
 using System;
namespace  komal
 { class  a { }
  struct  b { }
  interface  c{ }
  delegate int YourDelegate();
 enum  d{ }
 namespace  k2  {}
 class maincalss
 { static void Main(string[] args) { //Your program starts here... } } }
```

Data types

- C# is strongly typed language.
- Each variable must have a type.
- Data types must be declared when we declare the variable.
- These are categorized into three:

  Value Types

  Reference Types

  Pointer types
- Value Types : Predefined – Next slide

  UDT - Structures, Enumerations
- Pointers
- Reference : Predefined : Objects, Strings

  UDT : Arrays, Classes, Interfaces

- Variables that are based on value types directly contain values.
- Assigning one value type variable to another copies the contained value.
- This differs from the assignment of reference type variables, which copies a reference to the object but not the object itself.
- All value types are derived implicitly from the System.ValueType
- Unlike with reference types, you cannot derive a new type from a value type.
- A value type cannot contain the null value.
- Each value type has an implicit default constructor that initializes the default value of that type.

```
        int  a = new int();  same as int  a=0;
```

- All of the simple types are aliases of the .NET Framework System types. For example, int is an alias of System.Int32

```
int x = 123;                    c# Syntax
System.Int32  x = 123; CTS Syntax
```

C# Pre-defined Value Types

C# Type	.Net Framework Type	Signed	Bytes
sbyte	System.sbyte	Yes	1
short	System.Int16	Yes	2
int	System.Int32	Yes	4
long	System.Int64(L or l)	Yes	8
byte	System.Byte	No	1
ushort	System.Uint16	No	2
uint	System.Uint32	No	4
ulong	System.Uint64	No	8
float	System.Single(F or f)	Yes	4
double	System.Double(D or d)	Yes	8
decimal	System.Decimal(M or m)	Yes	12
char	System.Char	N/A	2
bool	System.Boolean	N/A	1

Difference between Value and Reference type

Value Types	Reference Types
Allocated on Stack	Allocated on Heap
Variable contains the value itself	Variable contains the address of the memory location where data is actually stored.
When copy, actual data is copied and each variable can be independently manipulated	Only memory address is copied.
Integer,float , double, boolean	String and object type
Struct is value type	Classes and interfaces are reference type

Literals

- Literals are a sequence of characters, digits and other characters.
- A literal can be used anywhere a value of its type is allowed.
- Integer literal : 123, 890,045,0x56
- Character : enclosed in a single quotes .Ex: ‘g’,’j’
- String : enclosed in double quotes. “k” , “komal”
- Floating point : 56.7f
- Boolean : true , false
- Long = 100l or 100L(UL is for unsigned)
- Double : 10.19 d or 10.19 D
- Decimal : 9.95m or 9.95M
- Integer literal is either of type int, uint, long or ulong

Variables

- An object stores its state in variables.
- A variable is an item of data named by an identifier.
- Identifiers are used for naming the namespaces, classes,methods, variables etc…
- You must explicitly provide a name and a type for each variable and optional initializer.
- The variable's name must be a legal identifier contains.
- The variable's type determines what values it can hold and what operations can be performed on it.
- To give a variable a type and a name, you write a variable declaration , which generally looks like this:
- type name ;

```
int a=10,  b =10;
int x = 123;
System.Int32 x = 123;(Framework type)
Console.WriteLine(x.GetType());
```

- We can use typeof operator also
- A variable has scope also.

- Variables can be initialized dynamically.

```
double s = a+ b;
```

- Every variable must have a data type. A variable's data type determines the values that the variable can contain and the operations that can be performed on it.
- Identifiers must not begin with digit.
- They can include alphabets, digits and underscore but whitespace is not allowed.
- Predefined keywords cannot be used.If we want to use, prefix that with @symbol.
- Lowercase and uppercase are different in C#.

Implicitly Typed variables:

- Let the compiler determine the type of a local variable based on the value used to initialize it. It is called an implicitly typed variable.
- This is declared using var and it must be initialized.
- Compiler uses the type of the initializer to determine the type of the variable.

  var e = 2.13f;

- Once the type has been determined , the variable has a type and this type is fixed throughput the lifetime of the variable.
- It is mainly used in LINQ.
- Only one implicitly typed variable can be declared at one time.

var s1=4.0, s2= 3;------→ It shows error

Default Values

bool	false
byte	0
char	'\0‘
decimal	0.0M
double	0.0D
float	0.0F
int	0
long	0L
sbyte	0
short	0
uint	0
ulong	0
ushort	0
struct	The value produced by setting all value-type fields to their default values and all reference-type fields to null.

Operators

- An operator performs a function on one, two, or three operands. An operator that requires one operand is called a *unary operator*.

  ++ is a unary operator that increments the value of its operand by 1.

- An operator that requires two operands is a binary operator

  a + b

- A *ternary operator* is one that requires three operands.

  op1 ? op2 : op3

Arithmetic Operators

- The following table summarizes the binary arithmetic operations in the c# programming language.

Relational and Conditional Operators

- A relational operator compares two values and determines the relationship between them. For example, != returns true if the two operands are unequal.

Shift and Logical Operators

- A shift operator performs bit manipulation on data by shifting the bits of its first operand right or left. This table summarizes the shift operators available in the c# programming language.
- Assignment Operators
- Other Operators

Operators

Control Flow Statements

- When you write a program, you type statements into a file.
- Without control flow statements, the interpreter executes these statements in the order they appear in the file from left to right, top to bottom.
- You can use control flow statements in your programs to conditionally execute statements, to repeatedly execute a block of statements, and to otherwise change the normal, sequential flow of control.

- while (*expression*) { *statement* }
- do { *statement(s)* } while (*expression*);
- for (*initialization*; *termination*; *increment*) { *statement* }
- if (*expression*) { *statement(s)* }
- if (*boolean expression*) { *statement(s)* } else { *statement(s)* }
- if (*boolean expression*) { *statement(s)* } else if (*boolean expression*) { *statement(s)* } else if (*boolean expression*) { *statement(s)* } else { *statement(s)* }
- switch (*integer expression*) { case *integer expression*: *statement(s)* break; ... default: *statement(s)* break; }

Statement Type	Keyword
looping	while, do-while , for ,foreach (Arrays)
Decision making	switch case, if –else
Jump	Break,continue, goto ,default

Note : goto identifier;
goto case constant expression;
goto default;

Constants

- The const keyword is used to modify a declaration of a field or local variable.

```
const int a=10;
```

- Only the C# built-in types (excluding System.Object) may be declared as const.
- It specifies that the value of the field or the local variable is constant, which means it cannot be modified.
- A constant expression is an expression that can be fully evaluated at compile time.
- User-defined types, including classes, structs, and arrays, cannot be const.(read only modifier)
- The constant declaration can declare multiple constants.

```
public const double x = 1.0, y = 2.0, z = 3.0;
```

- A constant can use in a constant expression

```
public const int a = 10; public const int b = a + 100;
```

- A const field can only be initialized at the declaration of the field.
- Constants are immutable values which are known at compile time and do not change for the life of the program
- c# does not support const methods, properties, or events.
- The enum type enables you to define named constants for integral built-in types.
- Constants can be marked as public, private, protected, internal, or protected internal .
- Static keyword cannot be used with the constants.
- Expressions that are not in the class that defines the constant must use the class name, a period, and the name of the constant to access the constant.

Conversion Rules

Implicit conversions:

- No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.
- For built-in numeric types, an implicit conversion can be made when the value to be stored can fit into the variable without being truncated or rounded off.
- There are no implicit conversions to the char type(*).
- There are no implicit conversions between floating-point types and the decimal type.

- **Explicit conversions (casts):**
- Explicit conversions require a cast operator.
- Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons.
- Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.
- The explicit numeric conversion may cause loss of precision or result in throwing exceptions.
- When you convert a decimal value to an integral type, this value is rounded towards zero to the nearest integral value.
- When you convert from a double or float value to an integral type, the value is truncated.
- When you convert double to float, the double value is rounded to the nearest float value. If the double value is too small or too large to fit into the destination type, the result will be zero or infinity.
- If the resulting value is outside the range of the destination type, an OverFlowEXception is thrown.

Arrays

- An array is a data structure that contains several variables of the same type.
- An array can be Single-Dimensional, Multidimensional or Jagged.
- The default value of numeric array elements are set to zero, and reference elements are set to null.
- A jagged array is an array of arrays, and therefore its elements are reference types and are initialized to null.
- Arrays are zero indexed: an array with n elements is indexed from 0 to n-1.
- Array elements can be of any type, including an array type.
- Array types are reference types derived from the abstract base type Array.
- We can use foreach iteration on all arrays in C#.

- Declaring Arrays: int[] numbers; int[,] numbers; int[][] numbers;
- Creation of the Arrays:

```
int[] numbers = new int[5];   single dimension
 int[,]  numbers = new int[5,4]; dimension
  int[][]  numbers = new int[5][];  Jagged
```

- Initialization:

```
int[] numbers = new int[5] {1, 2, 3, 4, 5};
int[] numbers = new int[] {1, 2, 3, 4, 5};
int[] numbers =  {1, 2, 3, 4, 5};
```

- Accessing array members is same as C and C++ using the index.

  numbers[5]
- length can be used to find the number of elements in a array.

Jagged Arrays

- A jagged array is an array whose elements are arrays.
- The elements of a jagged array can be of different dimensions and sizes.
- A jagged array is sometimes called an "array of arrays."
- A jagged array is an array of arrays, and therefore its elements are reference types and are initialized to null.
- The method Length returns the number of arrays contained in the jagged array.

Creation

```
int[][]  num  = new int[3][];
```

Initialization

num[0] = new int[5]

num[1] = new int[4];

num[2] = new int[2];

```
num[0] = new int[] { 1, 3, 5, 7, 9 };
num[1] = new int[] { 0, 2, 4, 6 };
num[2] = new int[] { 11, 22 };
```

- Two dimensional Jagged array:

int[][,] = new int[3][,];

Enumerations

- An enumeration is a set of named integer constants.It inherits from **System.Enum**

  enum name{list};

- The enum keyword is used to declare an enumeration, a distinct type that consists of a set of named constants called the enumerator list.
- We can define an enum directly within a namespace so that all classes in the namespace can access it.
- But it can be defined within a class or struct but not in method.
- By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1
- Enumerators can use initializers to override the default values.

Ex

- enum Months {Jan, Feb, Mar, Apr, June, July, Aug,Sep,Oct,Nov,Dec};
- enum Months {Jan=1, Feb, Mar, Apr, June, July, Aug, Sep,Oct,Nov,Dec};
- Members can be accessed thro their typename via the dot operator

- Every enumeration type has an underlying type, which can be any integral type except char.
- The default underlying type of enumeration elements is int.
- The approved types for an enum are byte, sbyte, short, ushort, int, uint, long, or ulong.
- If u want to change the type

  enum Months: byte {Jan=1, Feb, Mar, Apr, June, July, Aug,Sep,Oct,Nov,Dec};
- We can assign any values within the range of that type.
- An enumerator cannot contain white space in its name.
- The underlying type specifies how much storage is allocated for each enumerator
- An explicit cast is necessary to convert from enum type to an integral type.

  As per previous example :

  Console.WriteLine(Months.Mar + “ has the value” + (int) Months.Mar)
- We can use this with switch statement also. case Months.Jan

Boxing and Unboxing

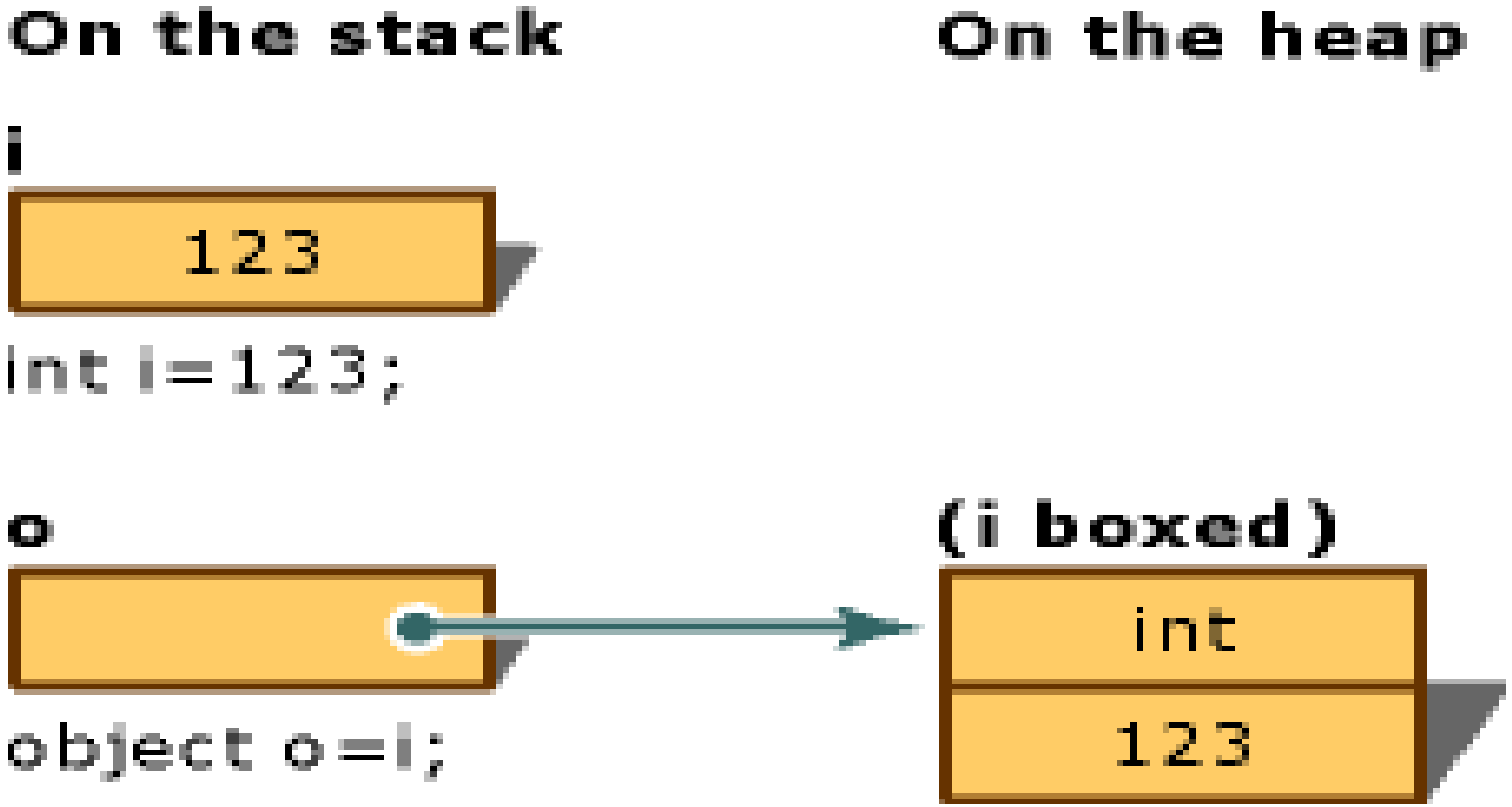
- Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type.
- When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap.
- Unboxing extracts the value type from the object.
- Boxing is implicit; Unboxing is explicit.

```
int i = 123;        object o = i;
```

- The object o can then be unboxed and assigned to integer variable i:

```
o = 123;   i = (int)o;
```

- Boxing is used to store value types in the garbage-collected heap.
- Boxing a value type allocates an object instance on the heap and copies the value into the new object.
-

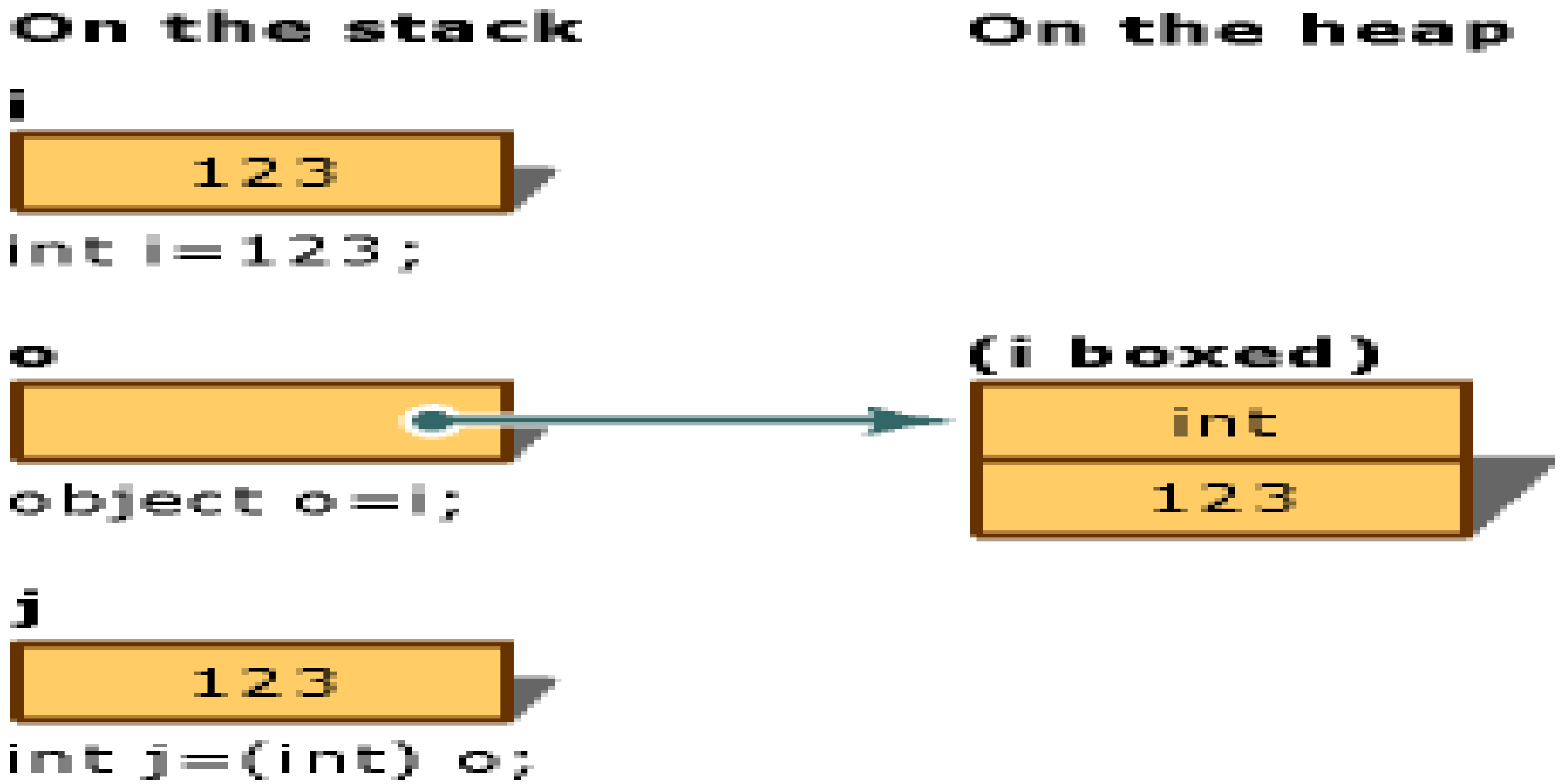

On the stack
On the heap
i
123
int i=123;
o
object o=i;
(i boxed)
int
123

UnBoxing

- Unboxing is an explicit conversion from the type object to a value type or from an interface type to a value type that implements the interface.
- An unboxing operation consists of:

  Checking the object instance to make sure that it is a boxed value of the given value type.

  Copying the value from the instance into the value-type variable.
- Attempting to unbox a reference to an incompatible value type causes an InvalidCastException.



```
int i=123;
object o=i;
int j=(int) o;
```