**Runtime Type ID**

* Runtime type ID is the mechanism that lets you identify a type during the execution of a program.
* Reflection is the feature that enables you to obtain information about a type.
* Using this information, you can construct and use objects at runtime.
* This feature is very powerful because it lets a program add functionality dynamically, during execution.
* An attribute describes a characteristic of some element of a C# program.
* For example, you can specify attributes for classes, methods, and fields, among others.
* Attributes can be interrogated at runtime, and the attribute information obtained.
* Attributes use both runtime type identification and reflection.

**Runtime Type Identification**

* Runtime type identification (RTTI) allows the type of an object to be determined during program execution.
* RTTI is useful for many reasons. For example, you can discover precisely what type of object is being referred to by a base-class reference. Another use of

RTTI is to test in advance whether a cast will succeed, preventing an invalid cast exception.

* Runtime type identification is also a key component of reflection.
* C# includes three keywords that support runtime type identification: is, as, and typeof.
* Each is examined in turn.

**Testing a Type with is:**

* You can determine if an object is of a certain type by using the is operator.
* Its general form is shown here:

expr is type

* Here, expr is an expression that describes an object whose type is being tested against type.
* If the type of expr is the same as, or compatible with, type, then the outcome of this operation is true. Otherwise, it is false. Thus, if the outcome is true, expr is some form of type.
* As it applies to is, one type is compatible with another if both are the same type, or if a reference, boxing, or unboxing conversion exists.

Here is an example that uses is:

// Demonstrate is.

using System;

class A {}

class B : A {}

class UseIs

{

static void Main()

{

A a = new A();

B b = new B();

if(a is A)

Console.WriteLine("a is an A");

if(b is A)

Console.WriteLine("b is an A because it is derived from A");

if(a is B)

Console.WriteLine("This won’t display -- a not derived from B");

if(b is B)

Console.WriteLine("B is a B");

if(a is object)

Console.WriteLine("a is an object");

}

}

**Using as:**

* Sometimes you will want to try a conversion at runtime, but not throw an exception if the conversion fails (which is the case when a cast is used).
* To do this, use the as operator, which has this general form:
  + expr as type
* Here, expr is the expression being converted to type.
* If the conversion succeeds, then a reference to type is returned. Otherwise, a null reference is returned.
* The as operator can be used to perform only reference, boxing, unboxing, or identity conversions.
* The as operator offers a streamlined alternative to is in some cases.
* For example, consider the following program that uses is to prevent an invalid cast from occurring:

// Use is to avoid an invalid cast.

using System;

class A {}

class B : A {}

class CheckCast

{

public static void Main(String[] args)

{

A a = new A();

B b = new B();

// Check to see if a can be cast to B.

if(a is B) // if so, do the cast

b = (B) a;

else // if not, skip the cast

b = null;

if(b==null)

Console.WriteLine("The cast in b = (B) a is NOT allowed.");

else

Console.WriteLine("The cast in b = (B) a is allowed");

}

}

**Using typeof:**

* Although useful in their own ways, the **as** and **is** operators simply test the compatibility oftwo types.
* Often, you will need to obtain information about a type. To do this, C# supplies the **typeof** operator.
* It retrieves a **System.Type** object for a given type. Using this object, you can determine the type’s characteristics.
* The **typeof** operator has this general form:

typeof(*type*)

* Here, *type* is the type being obtained. The **Type** object returned encapsulates the information associated with *type.*
* Once you have obtained a **Type** object for a given type, you can obtain information about it through the use of various properties, fields, and methods defined by **Type**.
* **Type** is a large class with many members, and a discussion is deferred until the next section, where reflection is examined.

using System.IO;

class UseTypeof

{

public static void Main()

{

Type t = typeof(StreamReader);

Console.WriteLine(t.FullName);

if(t.IsClass)

Console.WriteLine("Is a class.");

if(t.IsAbstract)

Console.WriteLine("Is abstract.");

else

Console.WriteLine("Is concrete.");

}

}