Microsoft

LESSON 6.2

10754 Microsoft .NET Fundamentals

Understand Type Safety

LESSON 6.2

Lesson Overview

Understand type safety.

In this lesson, you will learn about:

- Type safety and verification.
- The benefits of strong types and the Common Type System (CTS).
- Security policies.

Guiding Questions

1. What is type safety?
2. What are the advantages of using strong types?
3. What are security policies?

Anticipatory Set

Discuss with a partner:

Why should you make the fields (or instance variables) private when designing and implementing a class in an object-oriented programming language?

Type Safety

- Code is considered “type-safe” if it accesses only the memory locations that it is authorized to access.
  - Example: type-safe code cannot read or write values in another object’s private fields. It accesses types only in well-defined, allowable ways.
- Type safety is not mandatory, but it provides the following important security benefits:
  - When code is type-safe, the Common Language Runtime (CLR) can run assemblies in isolation, so they do not affect one another adversely.
  - Type-safe code cannot call unmanaged code without permission, which helps prevent malicious code from doing any damage.

Type Safety Verification

- Type safety is checked in the Just-In-Time (JIT) compilation process.
- The JIT compiler checks the assembly's metadata and the Microsoft® Intermediate Language (MSIL) code for the method to be compiled; it verifies that the code is type-safe.
- This verification is optional and will be bypassed if the code has been granted `SecurityPermission` with the passed enum member `SkipVerification`.
- Even though code may in fact be type-safe, not all compilers are able to generate verifiably type-safe code.
  - Microsoft C++ is an example of a compiler that cannot generate type-safe managed code that is verifiable at run time.
  - A tool called PEVerify.exe can determine if code is verifiably type-safe.
- Note: To benefit from code access security, assemblies must be verifiably type-safe!

Unsafe Code in C#

- "Unsafe" or unverifiable C# code can be executed in a context that is defined as unsafe.
  - Such code is not necessarily dangerous; it is just code whose safety cannot be verified by the CLR. The CLR, therefore, will execute unsafe code only if it is in a fully trusted assembly.
  - If you use unsafe code, it is your responsibility to ensure that your code does not introduce security risks or pointer errors.
- An unsafe context can be defined using the `unsafe` keyword:

```
unsafe
{
    // Unsafe context: can use unsafe code here.
}
```

- Because C# does not support pointer arithmetic, an unsafe context must be created to use pointers.

Strong Types

- C# is a strongly typed language. Every variable and constant has a type, as does every expression that evaluates to a value.
- Microsoft Visual Basic® allows undeclared variables and variables without a type, so it is not a strongly typed language.
  - Example: If a variable is used without declaring a type—or without declaring the variable at all—the compiler will assign a type implicitly.
- It is usually best to use strong typing by specifying data types for all variables.

Benefits of Strong Typing

- It enables Microsoft IntelliSense® support for variables and objects.
  - This allows the developer to see the properties and other members as the code is entered.
- It takes advantage of compiler type checking.
  - This catches statements that can fail at run time due to errors such as overflow. It also catches calls to methods on objects that do not support them.
- It results in faster execution of code.

Common Type System (CTS)

- The CTS defines how types are declared, used, and managed in the CLR.
  - This is an important part of the run time's support for cross-language integration.
- The CTS performs the following functions:
  - Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution.
  - Provides an object-oriented model that supports the complete implementation of many programming languages.
  - Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
  - Provides a library that contains the primitive data types (such as `Boolean`, `Byte`, `Char`, `Int32`, and `UInt64`) used in application development.

Security Policies

- Security policy is the configurable set of rules that the CLR follows when it decides what it will allow code to do.
- Administrators set security policy and the run time enforces it.
- The run time helps ensure that code can access only the resources—and call only the code—allowed by security policy.
- Whenever an attempt is made to load an assembly, the run time uses security policy to determine which permissions to grant to the assembly.

Think-Pair-Share

1. Independently think of a response to each question below.
2. Pair up with your partner and discuss your answers and resolve differences.
3. Share your responses with the group.

Questions:

- Why is type safety important?
- Microsoft Visual C#® is a “strongly typed” language—what does this mean?
- List benefits of using strong types.