Creating Classes and Implementing Type-Safe Collections

Creating Classes and Members

- Use the **class** keyword

```
public class DrinksMachine
{
   // Methods, fields, properties, and events.
}
```

- Specify an access modifier:
  - public
  - internal
  - private
- Add methods, fields, properties, and events

The Main Method

Reference Types and Value Types

- Value types
  - Contain data directly

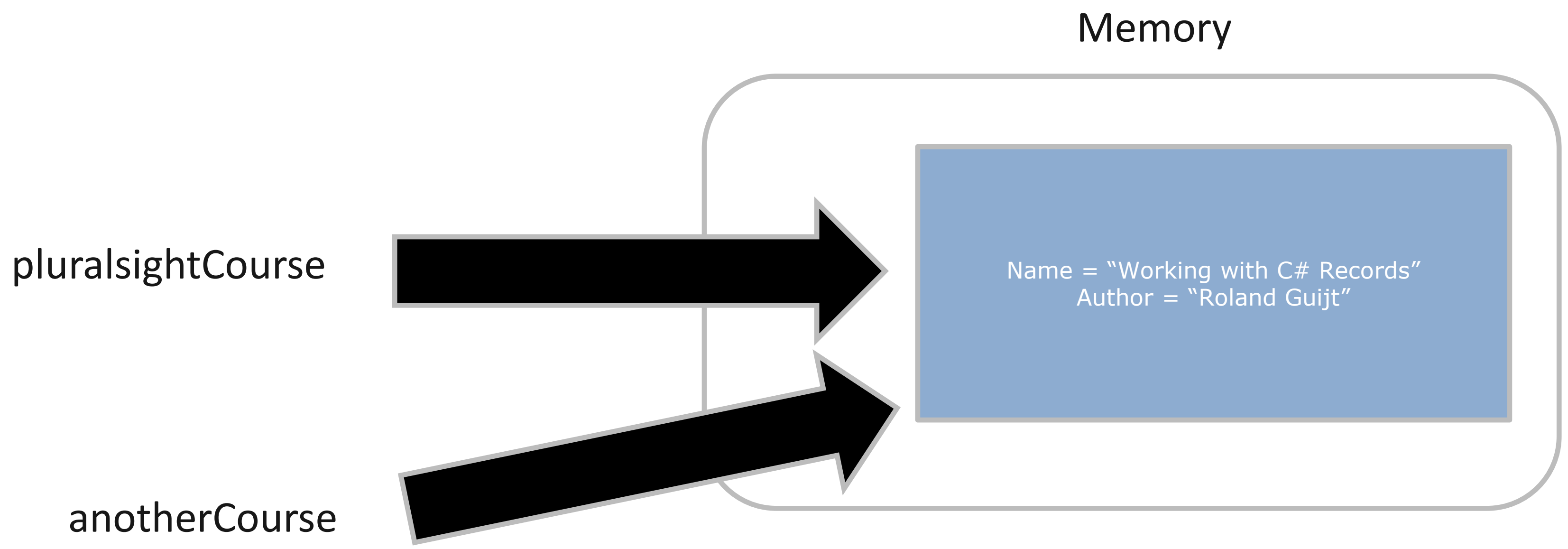
```
int First = 100;
int Second = First;
```

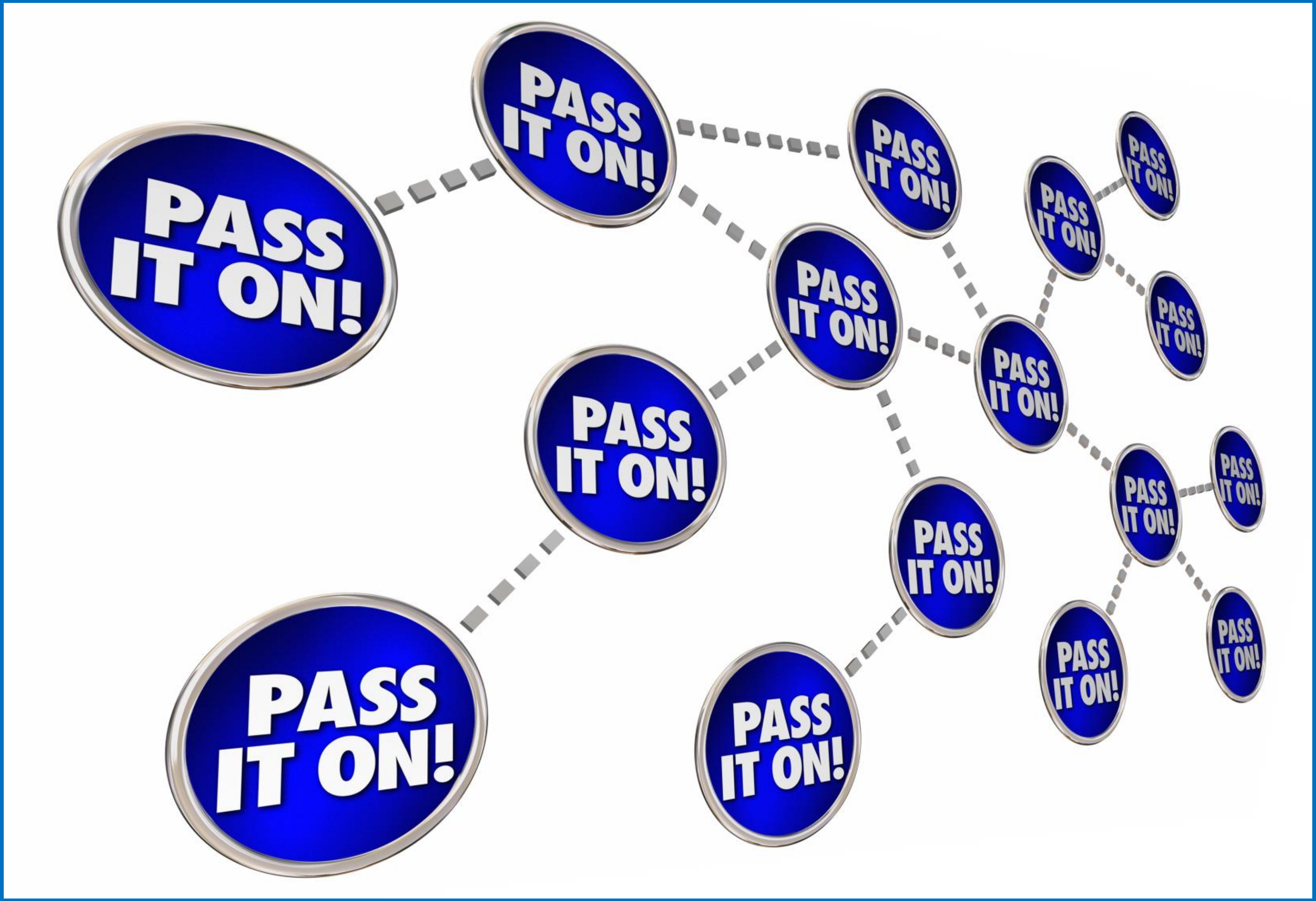
  - In this case, **First** and **Second** are two distinct items in memory
- Reference types
  - Point to an object in memory

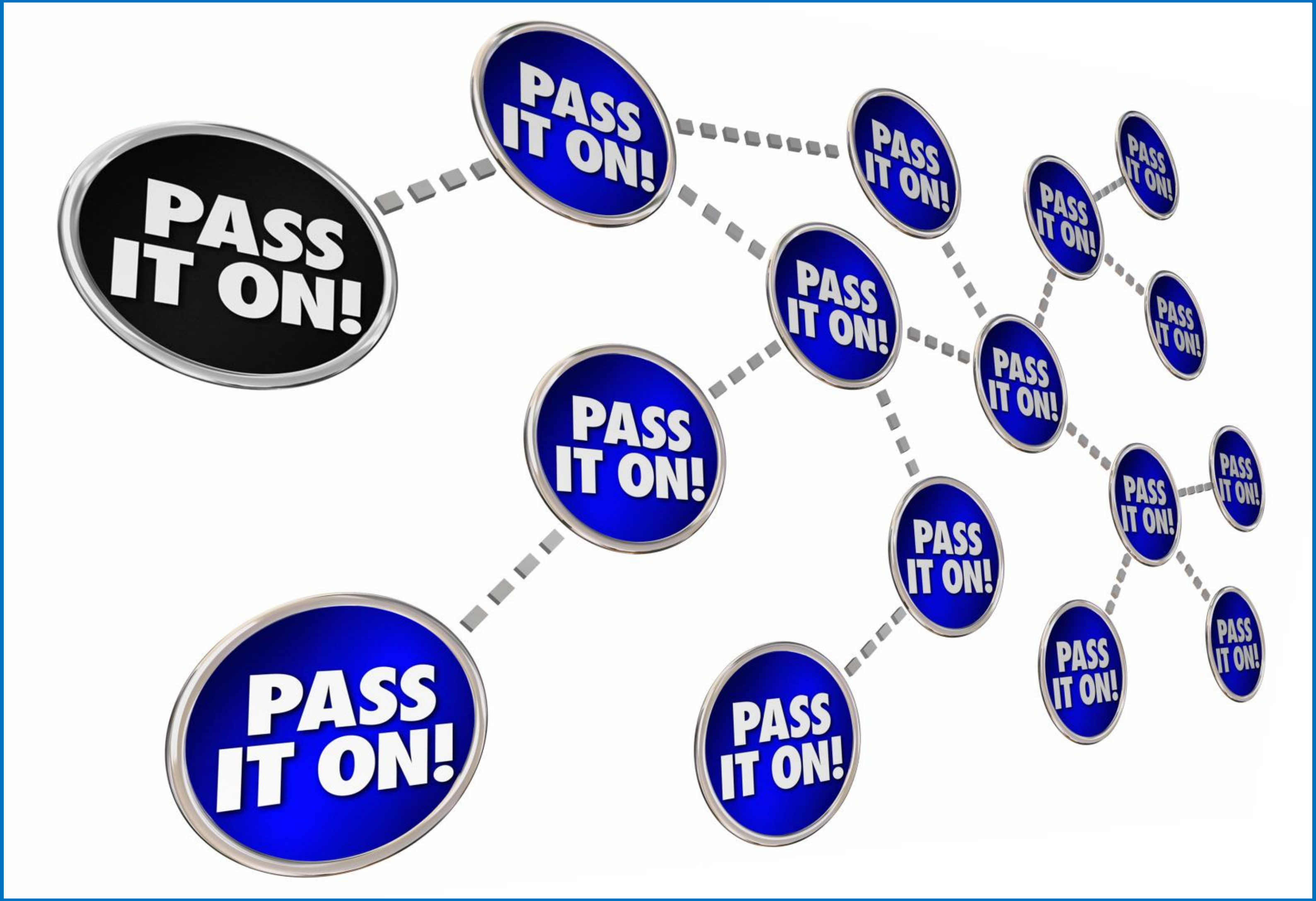
```
object First = new Object();
object Second = First;
```

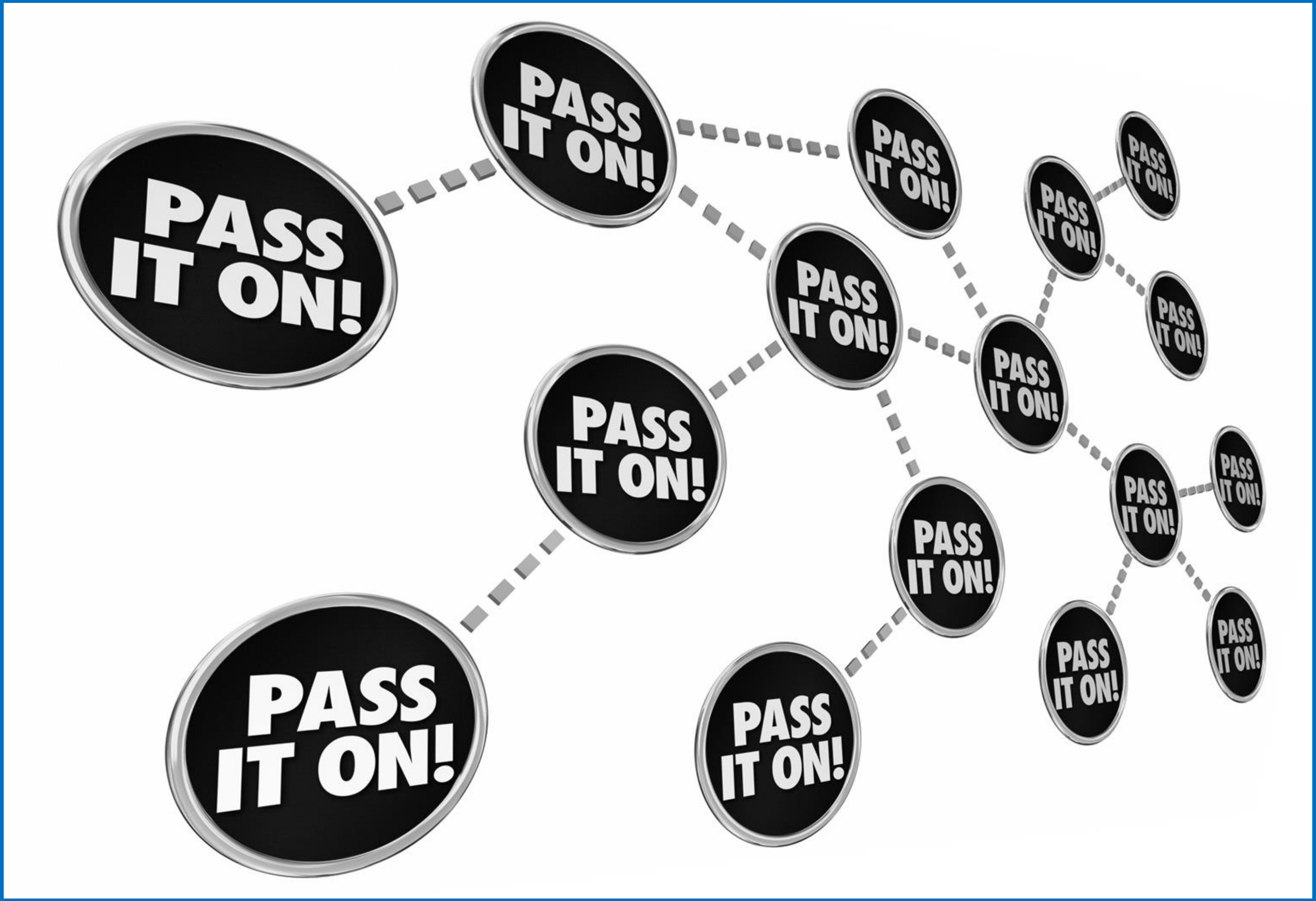
  - In this case, **First** and **Second** point to the same item in memory

Reference Types




PASS IT ON!


PASS IT ON!


PASS IT ON!

Instantiating Classes

- To instantiate a class, use the **new** keyword

```
DrinksMachine dm = new DrinksMachine();
```

- To infer the type of the new object, use the **var** keyword

```
var dm = new DrinksMachine();
```

- To call members on the instance, use the dot notation

```
dm.Model = "BeanCrusher 3000";
dm.Age = 2;
dm.MakeEspresso();
```

Using Constructors

- Constructors are a type of method:
  - Share the name of the class
  - Called when you instantiate a class
- A default constructor accepts no arguments

```
public class DrinksMachine
{
   public DrinksMachine()
   {
      // This is a default constructor.
   }
}
```

- Classes can include multiple constructors
- Use constructors to initialize member variables

Using Constructors

- Constructors are a type of method:
  - Share the name of the class
  - Called when you instantiate a class
- A default constructor accepts no arguments

```
public class DrinksMachine
{
   public DrinksMachine()
   {
      // This is a default constructor.
   }
}
```

- Classes can include multiple constructors
- Use constructors to initialize member variables

Practicing classes

- Create a class (be creative ☺)
- Instantiate it
- Use it

Lab 4

Creating an object-oriented version of Snakes and Ladders.

Thinking in Classes: Abstraction

- When beginning a task, immediately think in classes. They are the building blocks for every C# application.
- “A customer has to place trades” => Customer and Trade classes
- “A game has a player and a gameboard with squares” => Player, GameBoard and Square classes.

Classes vs. Structs

- Generally, classes are used to build an application
- Passing around object references is cheaper than passing around value types
- Structs often used for simple data structures. Example: DateTime
- Structs lack object orientation features like inheritance

Creating Static Classes and Members

- Use the static keyword to create a static class

```
public static class Conversions
{
   // Static members go here.
}
```

- Call members directly on the class name

```
double weightInKilos = 80;
double weightInPounds =
   Conversions.KilosToPounds(weightInKilos);
```

- Add static members to non-static classes

Namespaces

- .NET contains numerous libraries
- To avoid clashes, we use namespaces
  - Grouping of related types
  - One or more assemblies

```
namespace YourCompanyName.BookingSystem
{
    class Customer
    {
        //...
    }
}
```

```
namespace SomeExistingCrmSystem
{
    class Customer
    {
        //...
    }
}
```

  - System.IO: file-related
  - System.Data: database types
- One assembly can contain one or more namespaces
  - Most of the time, they contain MANY!
- If a namespace is not explicitly supplied, then the type will be added to a nameless global namespace

Specifying a Namespace

- By prefixing the typename
  - Allows using classes with same name from different namespaces

```
YourCompanyName.BookingSystem.Customer a;

SomeExistingCrmSystem.Customer b;
```

- The "using" directive

```
using YourCompanyName.BookingSystem;

class MyClass
{
    Customer a;
}
```

Namespace Features

- Use File-based namespaces to reduce nesting
- Create folders in your project to use sub namespaces

The System namespace

- System is the most important namespace in .NET
  - Core functionality of .NET
  - System.Int32
  - System.String
  - …

Referencing External Assemblies

- Create other assemblies to reuse or separate functionality
- Reference is in the metadata of the assembly
- The internal accessor is used to limit visibility of types to the current assembly

Using the NuGet Package Library

Garbage Collector

```
class Customer
{
    public string Name;
}

class MainClass
{
    public void test()
    {
        Customer c = new Customer();
        c.Name= "George";
   }
}
```

- **You don't know when the Garbage Collector will clean the heap**

Demonstration: Other Class Features

- this
- Access modifiers
- Object initializers
- Const and readonly
- Partial classes