**Generische Klassen in C# – Erstellung und Anwendung**

Vor der Einführung von **Generics** musste man für verschiedene Datentypen eigene Klassen erstellen oder mit object arbeiten.

Die Nachteile waren: Redundanz, Code-Duplikation, Verlust der Typsicherheit.

**Beispiel ohne Generics** (als separate Klassen):

**Schlecht**, weil aufwendig in der Herstellung, Code-Redundanz (Copy-Paste-Programmierung)

public class IntegerBox

{

public IntegerBox(**int** content)

{

Content = content;

}

public **int** Content { get; set; }

}

public class StringBox

{

public StringBox(**string** content)

{

Content = content;

}

public **string** Content { get; set; }

}

**Beispiel ohne Generics** (mit fixem Datentyp Object):

**Schlecht** wegen schlechter Performance, Typunsicherheit und anschließender Unboxing

var box1 = new Box(2);

// zu welchem Typen soll ich es unboxen!?

var a = (int)box1.Content;

var b = (string)box1.Content;

var c = (List<double>)box1.Content;

**Beispiel mit Generics:**

**Gut**, weil wiederverwendbar und typsicher.

**T** (oder ein anderer Buchstabe) fungiert als Platzhalter für den Typ, den ich in dieser Klasse verwenden möchte.

public class Box<**T**>

{

public Box(**T** content)

{

Content = content;

}

public **T** Content { get; set; }

}

**Merke:**

Generics erlauben dem Programmierer eine **typsichere Verwendung von Klassen, die nicht an einen bestimmten Typ gebunden sind**. Der Nutzer der Klasse gibt den (die) Datentyp(n) vor, die die Klassen verarbeiten soll.

**Einschränkungen (Constraints) für generische Klassen**

Manchmal reicht **T** alleine nicht, man möchte den **Typ einschränken**.

**Beispiel**:

public class Repository<T> where T : class

{

private List<T> items = new List<T>();

public void Add(T item)

{

items.Add(item);

}

public IEnumerable<T> GetAll()

{

return items;

}

}

**Wichtige Einschränkungen (Constraints):**

where T : struct → Nur Werttypen

where T : class → Nur Referenztypen

where T : new() → Muss einen parameterlosen Konstruktor haben

where T : InterfaceName → Muss ein bestimmtes Interface implementieren

Weitere Einschränkungsmöglichkeiten:

<https://learn.microsoft.com/de-de/dotnet/csharp/programming-guide/generics/constraints-on-type-parameters>

**Übung**

Erstellen Sie eine generische Klasse „Geldboerserl“, die nur Referenztypen und Typen mit parameterlosem Konstruktor akzeptiert.  
Die Klasse bietet zwei Methoden an: Eine, um eine Währung hinzufügen und eine um die Anzahl von Währungen zurückzugeben.  
Instanziieren Sie diese Klasse mit einem (vorher zu erstellenden) Typ und geben Sie die Anzahl der hinzugefügten Währungen in der Konsole aus.