Serialisierung
Binäre Serialisierung

Ingo Köster

Diplom-Informatiker (FH)

Serialisierung

- Speicherung von Objekten
  - In Dateien
  - Für die Übertragung von Objekten über das Netzwerk
    - Z.B. bei Verteilten Systemen

- Kompletter Zustand des Objektes, inklusive aller referenzierten Objekte, wird in einen Datenstrom umgewandelt

- Dieser wird anschließend z.B. auf ein Speichermedium geschrieben

Serialisierung

- Nach der Serialisierung liegt ein Objekt mehrfach vor
  - als externe Darstellung (z.B. als Datei)
  - im Arbeitsspeicher
- Nach der Serialisierung haben Änderungen am Objekt im Arbeitsspeicher keine Auswirkung auf das serialisierte Objekt
- Umkehrung der Serialisierung wird als Deserialisierung bezeichnet

Serialisierungsverfahren

- Objekte müssen in ein definiertes Format überführt werden, um für die Deserialisierung eine eindeutige Interpretation sicherzustellen
- Die Daten eines Objektes werden einem Bytestrom übergeben
- Verschiedene Klassen sind zur Serialisierung verwendbar
  - Überführen Objekte in Binär-, XML- und JSON-Daten

Serialisierungsverfahren - `BinaryFormatter`

- Klasse `BinaryFormatter`
  - Namensraum: `System.Runtime.Serialization.Formatters.Binary`
- Ideal, wenn Daten zwischen verschiedenen .NET Anwendungen ausgetauscht werden sollen
- Zu serialisierende Klassen müssen mit Attributen gekennzeichnet werden

Serialisierung

- Methoden
  - `Serialize` und `Deserialize`

- `public void Serialize(Stream, object)`
  1. Referenz auf ein Objekt vom Typ Stream
  2. Referenz auf das Objekt, welches serialisiert werden soll

- `public object Deserialize(Stream)`

Serialisierung – Attribute für BinaryFormatter

```
[Serializable()]
class Person {...}
```

› Alle Felder & Properties der Klasse `Person` werden serialisiert
  › unabhängig davon, ob sie `private` oder `public` sind

› Lokale Variablen und statische Klassenvariablen werden nicht serialisiert

› Die runden Klammern können weggelassen werden

```
[Serializable]
```

Beispiel Serialisierung

```
// Zu serialisierendes Objekt
Person p = new Person("Meyer",42);

FileStream stream = File.Create(@"c:\Person.dat");

BinaryFormatter formatter = new BinaryFormatter();

// Serialisierung
formatter.Serialize(stream, p);

stream.Close();
```

Beispiel Deserialisierung

```
// Referenz für das deserialisierte Objekt
Person diePerson;

FileStream stream = File.OpenRead(@"c:\Person.dat");

BinaryFormatter formatter = new BinaryFormatter();

// Deserialisierung
diePerson = (Person)formatter.Deserialize(stream);

stream.Close();
```

Auflistungen

- Es können mehrere Objekte in eine Datei serialisiert werden
  - Hinweis: Kein Anhängen von weiteren Objekten an Dateien möglich
- Es können Arrays und Collections serialisiert werden
- Besitzt ein zu serialisierendes Objekt Referenzen auf andere Objekte eines anderen Typs, müssen diese ebenfalls serialisierbar sein
  - Beispiel: Klasse `Person` hat Eigenschaft vom Typ `Adresse`
  - Attribut `[Serializable]` bei `BinaryFormatter`

Nichtserialisierbare Daten

- Alle Instanzfelder einer Klasse werden serialisiert
  - Nicht immer wünschenswert
- Instanzfelder, die nicht serialisiert werden sollen, können durch das Setzen des Attributs `NonSerialized` vor der Deklaration ausgeschlossen werden
  - Nicht für Properties verwendbar

```
[Serializable]
class Person {
  public string name;
  [NonSerialized]
  private int alter; // Wird nicht serialisiert
}
```

Vererbung

- Das Serializable-Attribut wird nicht vererbt
- Beispiel:
  - Serialisierbare Klasse `Person` und daraus die Klasse Kunde ableiten
- Attribut `Serializable` ist selbst dann in Kunde nötig, wenn nur die aus `Person` geerbten Mitglieder serialisiert werden sollen
- Sollen die Felder der Klasse Kunde, die aus `Person` abgeleitet ist, serialisiert werden, muss auch die Basisklasse serialisierbar sein