Konzepte objektorientierter Programmierung

– Lecture 12 –

Prof. Dr. Peter Müller

Chair of Programming Methodology

Herbstsemester 2008

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Summary

- Frame properties
  - Crucial for program verification
  - Difficult to specify and prove (abstraction)
  - No good solution for runtime assertion checking
- Invariants
  - Semantics of invariants is non-trivial
  - Handling callbacks is difficult, especially for runtime assertion checking
  - Invariants of object structures require strong encapsulation

Agenda for Today

12. Verification

Objectives

- Excitement

Spec# Approach

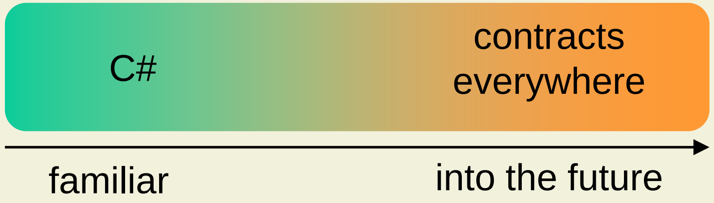
- Specifications record design decisions
    - Bridge intent and code
- Tools amplify human effort
    - Manage details
    - Find inconsistencies
    - Ensure quality

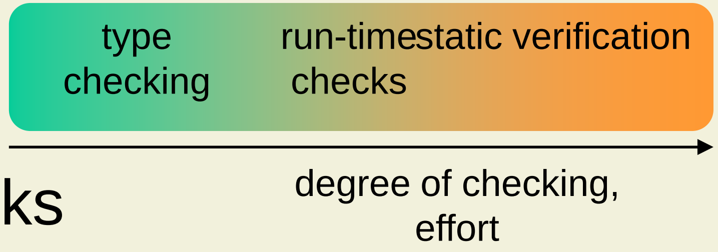
Goals of the Spec# Project

- Build the best such system we can build today
- Experiment with the system to get a feel for what it is like to use
- Advance the state of the art

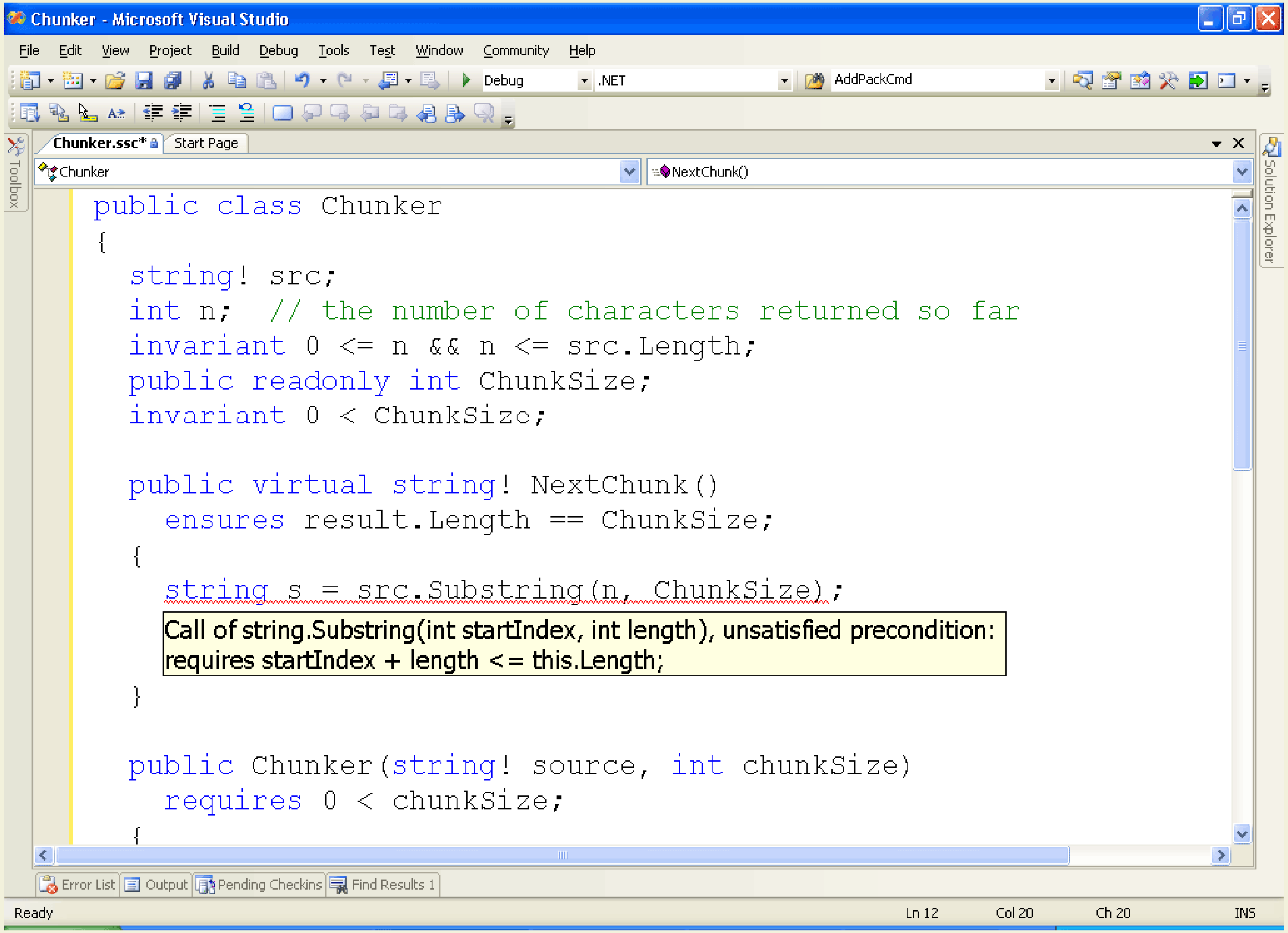
Spec#

- Experimental mix of contracts and tools
- Aimed at experienced developers who know the high cost of testing and maintenance
- Superset of C#
  - Non-null types
  - Contracts
- Tool support
  - More type checking
  - Compiler-emitted run-time checks
  - Static program verification





Spec# Demo



```
public class Chunker
{
  string! src;
  int n;  // the number of characters returned so far
  invariant 0 <= n && n <= src.Length;
  public readonly int ChunkSize;
  invariant 0 < ChunkSize;

  public virtual string! NextChunk()
    ensures result.Length == ChunkSize;
  {
    string s = src.Substring(n, ChunkSize);
  }

  public Chunker(string! source, int chunkSize)
    requires 0 < chunkSize;
  {
```

Call of string.Substring(int startIndex, int length), unsatisfied precondition: requires startIndex + length <= this.Length;

Spec# Demo

- Non-null types are simple and powerful
- Contract checking starts from existing libraries
- Verification is seen by the programmer as a kind of extended “type” checking
- “Tool tips” can use contracts as well as signature
- Verification works!

Spec# Verifier

- Verifier checks programs for coding errors …
    - Null-dereferences
    - Array bounds errors
    - Illegal casts

- … and specification violations
    - Simple pre-post specifications
    - Invariants

Program Checker Design Tradeoffs

- Objectives
    - Fully automated reasoning
    - As little annotation overhead as possible
    - Performance

- Spec# verifier is sound
    - No errors are missed

- Spec# verifier is not complete
    - Warnings do not always report errors (false alarms)

Demo: McCarthy’s Function

```
public static int f( int n )


{
  if ( n > 100)   return n – 10;
  else            return f( f( n + 11 ) );
}
```

Demo: Callbacks

```
class Purse {
  int amount;
  Currency! crncy;
  invariant 0 ≤ amount ∧
      ( crncy.symbol = Currency.CHF ⇒ amount % 5 = 0 );
  void Exchange( Currency! to ) {
    amount := crncy.Convert( amount, to );
    crncy := to;
    x.P( this );
    if ( crncy.symbol = Currency.CHF )  amount := amount / 5*5;
  }
… }
```

Demo: Multi-Object Invariants

```
class Person {
  Purse! purse;
  invariant  purse.amount % 100 = 0;
  … }
```

```
class Purse {
  void Exchange( Currency! to )
    requires inv = valid;
  {
    expose this {
      amount := crncy.Convert( amount, to );  …
    }
  } … }
```

Object Structures: assignment might break invariants of client objects