i219 Software Design Methodology
1. Overview of software design methodology

Kazuhiro Ogata (JAIST)

Outline of lecture

- Activities involved in software development
- Unified Modeling Language (UML) for specifying requirements/designs
- Java for implementing designs (writing programs for designs)
- Java Path Finder (JPF) for model checking Java multithreaded programs
- Outline of design & implementation of a simple calculator

Activities involved in software development

- Domain description and/or analysis
- Requirements specification and/or analysis
- Software design specification and/or analysis
- Implementation (programming)
- Analysis (testing, model checking, etc.) of programs
- Deployment
- Maintenance (or evolution)

Requirements/design specification

- Use of (structured) natural languages
  - Less systematic & informal
- Object-oriented design methodologies (OODM)
  - Systematic but informal
- Formal methods
  - Systematic and formal

A history of OODM in a nutshell (1)

- In the 1980s, Smalltalk became a stable platform and C++ was born.
- Between 1988 and 1992, several object-oriented graphical modeling languages were proposed. Among them are
  - OOAD by Grady Booch; OOA & OOD by Peter Coad
  - OOSE by Ivar Jacobson; OMT by Jim Rumbaugh
  - Responsibility Driven Design by Rebecca Wirfs-Brock
- In 1994, Rumbaugh (GE) joined Booch at Rational (now a part of IBM).
- By OOPSLA '95, Booch & Rumbaugh prepared the 1st public document of their merged method: version 0.8 of the Unified Method documentation.

A history of OODM in a nutshell (2)

- In 1995, Jacobson (Objectory) joined Booch & Rumbaugh.
- The OMG decided to take a major role for standardization because of interoperability.
- In Jan. 1997, Rational collaborated other organizations and released version 1.0 of the UML documentation.
- In Nov. 1997, the OMG adopted version 1.1 of the UML documentation as an official OMG standard.
- In 2005, UML 2.0 was released.
- In 2011, UML 2.4.1 was released.

UML in a nutshell

- 13 kinds of diagrams for structure & behavior
  - 6 kinds of diagrams for structure
    Eg., class diagram, object diagram
  - 7 kinds of diagrams for behavior
    E.g., use case diagram, sequence diagram, state diagram
- 3 ways of using UML as
  - Sketch
  - Blueprint
  - Programming language
- Can be used for 2 directions of engineering
  - Forward engineering
  - Backward engineering

Programming Paradigms

- Imperative (procedural) programming
  - Pascal, C, C++, Java, Lisp, Standard ML, Oz, Ruby, Python
- Logic programming
  - Prolog, Oz, GHC, KL1
- Functional programming
  - Miranda, Haskell, Erlang, Lisp, Standard ML, Oz, Scala, Maude, CafeOBJ
- *Object-oriented programming*
  - Smalltalk, C++, *Java,* Self, Oz, Scala, Ruby, Python, ABCL/1, ConcurrentSmalltalk, MultithreadSmalltalk

Java in a nutshell

- An object-oriented programming language Designed (developed) by James Gosling at Sun Microsystems (merged into Oracle Corporation) at 1995
- Inherited many concepts & technologies from Smalltalk
  - bytecode, virtual machine, garbage collection
  - multi-threads, dynamic (just-in-time) compilation
  - classes, inheritance, reflection, …
- Some differences from Smalltalk
  - statically typed, primitive data types such as `int` (values of such types such as `3` and `4` are not objects), …

Model checking

- A way to formally verify that a design or implementation (program) of software enjoys desired properties (requirements)
- Traditional model checking (for designs/programs)
  - Designs/programs are supposed to be modeled as state machines (mainly) by human users
  - Among such model checkers: NuSMV, Spin, SAL, PAT, Maude model checker
- Software model checking (for programs)
  - Programs can be model checked and do not need to be modeled as state machines by human users
  - Among such model checkers: Slam (for C programs), CBMC (for C/C++ programs), JPF (for Java programs)

Java Path Finder (JPF) in a nutshell

- A software model checker for Java programs developed at NASA Ames Research Center
- Originated from Klaus Havelund in 1999; initially implemented as a Java-to-Promela translator (with Spin as a model checker); currently JPF has its own virtual machine implemented in Java
- Not only a software model checker but also can do many others for Java programs; hence called the Swiss army knife for Java verification

Outline of design & implementation of a simple calculator

- Requirements in use cases
- A use case diagram
- A sequence diagram
- A class diagram
- An object diagram
- A piece of Java code
- Use of the calculator

Requirements in use cases (1)

Calculate Expression

Main Success Scenario:
1. User inputs an expression
2. System calculates the expression
3. System displays the result

Extensions:
2a: The input has a syntax error
.1: System displays this
3a: Division-by-zero occurs
.1: System displays this

Show Usage

Main Success Scenario:
1. User inputs the help command
2. System displays the usage

Quit

Main Success Scenario:
1. User inputs the quit command
2. System quits

Requirements in use cases (2)

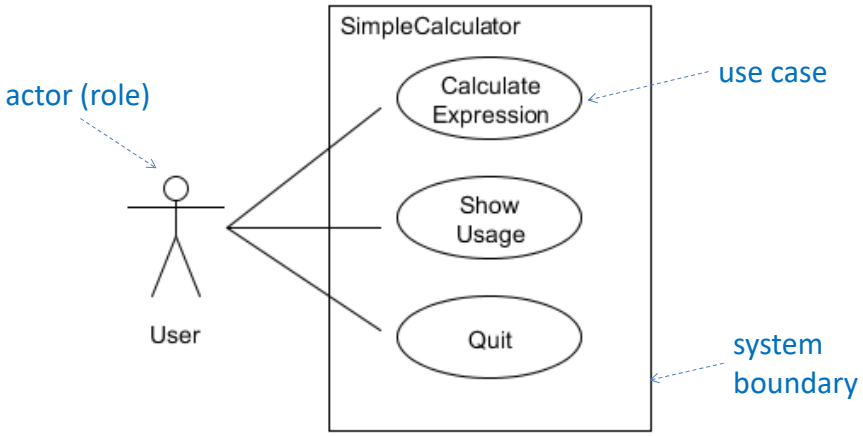
Calculate Expression in String

Main Success Scenario:
1. CLP gives a string to SC
2. SC transforms the string into a list of tokens
3. SC makes a parse tree from the list
4. SC interprets the parse tree and gets the result
5. SC returns the result

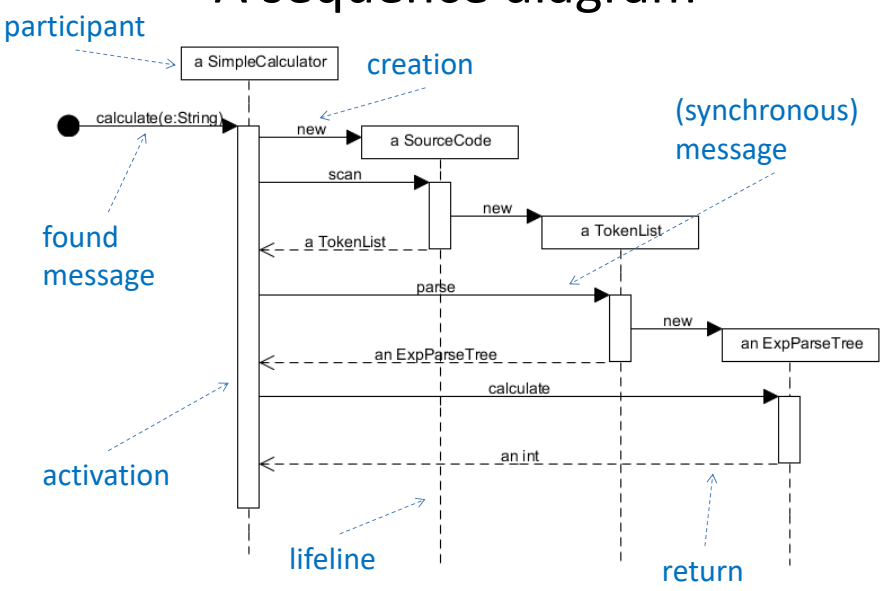
Extensions:
3a: A parse tree cannot be made from the list
.1: SC reports a syntax error
4a: Division-by-zero occurs
.1: SC reports the occurrence

CLP stands for Command Line Processor.
SC stands for Simple Calculator.

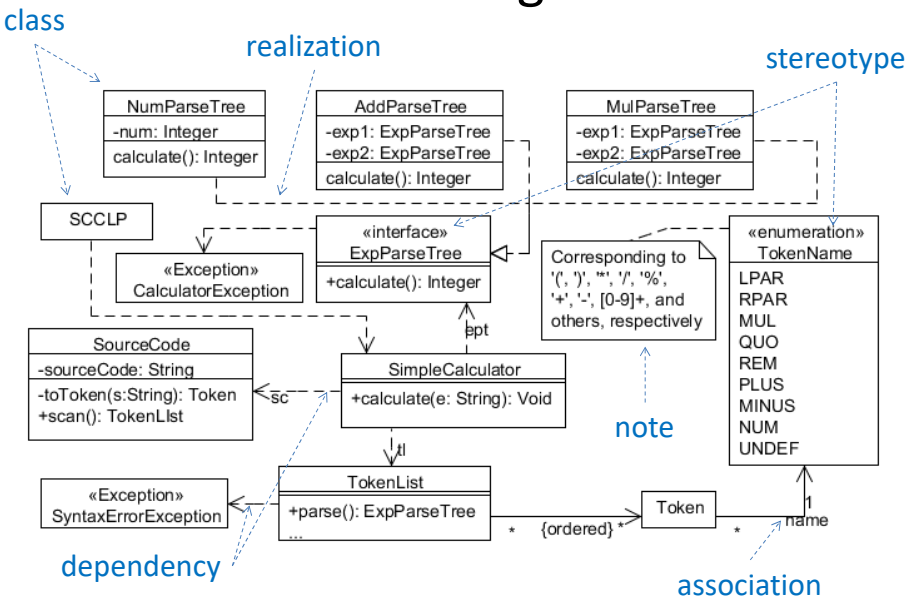
A use case diagram



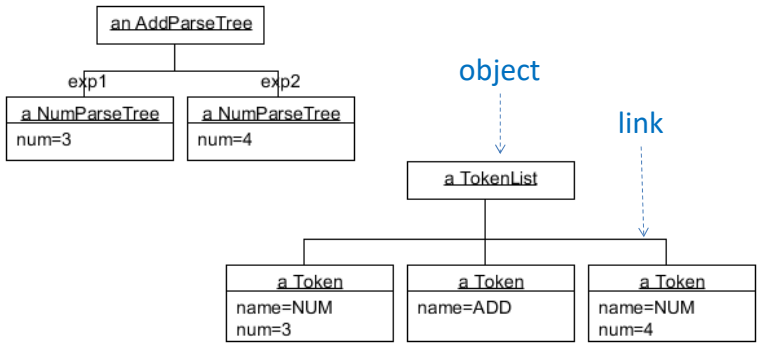
A sequence diagram



A class diagram



An object diagram



A piece of Java code

```
public class SimpleCalculator {
   public SimpleCalculator() {}
   public void calculate(String exp) {
      SourceCode sc = new SourceCode(exp);
      TokenList tl = sc.scan();
      try {  ExpParseTree ept = tl.parse();
         System.out.println(ept.calculate());
      } catch (SyntaxErrorException e) {
         System.err.println(e.getMessage());
      } catch (CalculatorException e) {
         System.err.println(e.getMessage());
      }
   }
   … }
```

A SourceCode sc is made

A list tl of tokens is made by sending scan() to sc

A parse tree ept is made by sending parse() to tl

A calculation is done by sending calculate() to ept and the result is displayed

Use of the calculator

```
$ java SCCLP
**********************
* Simple Calculator *
**********************
SimpleCal> 3+4
[num=3, +, num=4]
add(3,4)
7
SimpleCal> 3+4*5
[num=3, +, num=4, *, num=5]
add(3,mul(4,5))
23
SimpleCal>
```

Summary

- Activities involved in software development
- UML for specifying requirements/designs
- Java for implementing designs (writing programs for designs)
- JPF for model checking Java programs
- Outline of design & implementation of a simple calculator