Unit-4

Concurrency Exception Handling

CONCEPTS

Concurrency

Exception Handling

CONCEPTS

Introduction to logic programming language A Brief Introduction to Predicate Calculus Predicate Calculus and Proving Theorems An Overview of Logic Programming The Origins of Prolog

The Basic Elements of Prolog Deficiencies of Prolog Applications of Logic Programming

C++ Templated Classes

Classes can be somewhat generic by writing parameterized constructor functions.

```
stack (int size) {
stk_ptr = new int [size];
max_len = size - 1; top = -1;
}
stack (100) stk;
```

The stack element type can be parameterized by making the class a templated class.

---> SHOW the templated class stack .

- Java does not support generic abstract data types

Object Oriented Programming in **Smalltalk**

Type Checking and Polymorphism:

All bindings of messages to methods is dynamic.

The process is to search the object to which the message is sent for the method; if not found, search the superclass, etc.

Because all variables are typeless, methods are all polymorphic

Inheritance.

All subclasses are subtypes (nothing can be hidden).

All inheritance is implementation inheritance. No multiple inheritance.

Methods can be redefined, but the two are not related.

C++

General Characteristics: Mixed typing system.

Constructors and destructors.

Elaborate access controls to class entities.

Inheritance:

A class need not be subclasses of any class. Access controls for members are:

Private (visible only in the class and friends). Public (visible in subclasses and clients).

Protected (visible in the class and in subclasses).

- In addition, the subclassing process can be declared with access controls, which define potential changes in access by subclasses.
- Multiple inheritance is supported

Java

Dynamic Binding

In Java, all messages are dynamically bound to methods, unless the method is final.

Encapsulation

Two constructs, classes and packages.

Packages provide a container for classes that are related.

Entities defined without an scope (access) modifier have package scope, which makes them visible throughout the package in which they are defined

Every class in a package is a friend to the package scope entities elsewhere in the package.

Ada 95

Example:

```
with PERSON_PKG; use PERSON_PKG;
package STUDENT_PKG is
type STUDENT is new PERSON with record
GRADE_POINT_AVERAGE : FLOAT;
GRADE_LEVEL : INTEGER;
end record;
procedure DISPLAY (ST: in STUDENT); end
STUDENT_PKG;
```

- DISPLAY is being overriden from PERSON_PKG
- All subclasses are subtypes
- Single inheritance only, except through generics

Concurrency

Def: A thread of control in a program is the sequence of program points reached as control flows through the program.

Categories of Concurrency:

Physical concurrency - Multiple independent processors (multiple threads of control).

Logical concurrency - The appearance of physical concurrency is presented by timesharing one processor (software can be designed as if there were multiple threads of control).

- Coroutines provide only quasiconcurrency.

Reasons to Study Concurrency

- It involves a new way of designing software that can be very useful--many real-world situations involve concurrency.
- Computers capable of physical concurrency
- are now widely used.

Design Issues for Concurrency

How is cooperation synchronization provided?

How is competition synchronization provided?

How and when do tasks begin and end execution?

Are tasks statically or dynamically created?

Semaphores

Semaphores (Dijkstra - 1965).

A semaphore is a data structure consisting of a counter and a queue for storing task descriptors.

Semaphores can be used to implement guards on the code that accesses shared data structures.

Semaphores have only two operations, wait and release (originally called P and V by Dijkstra).

Semaphores can be used to provide both competition and cooperation synchronization

Example

wait(aSemaphore)

if aSemaphore's counter > 0 then

Decrement aSemaphore's counter

else

Put the caller in aSemaphore's queue

Attempt to transfer control to some ready task

(If the task ready queue is empty, deadlock occurs) end

Example

release(aSemaphore)

if aSemaphore's queue is empty then

Increment aSemaphore's counter

else

Put the calling task in the task ready queue

Transfer control to a task from aSemaphore's queue

end

Monitors

- Competition Synchronization with Monitors:
- Access to the shared data in the monitor is
  - limited by the implementation to a single process at a time; therefore, mutually exclusive access is inherent in the semantic definition of the

Monitors

- Cooperation Synchronization with Monitors:
  - Cooperation is still required - done with semaphores, using the queue data type and the built-in operations, delay (similar to send) and continue (similar to release).
  - delay takes a queue type parameter; it puts the process that calls it in the specified queue and removes its exclusive access rights to the monitor's data structure.
- Differs from send because delay always blocks the caller.
  - continue takes a queue type parameter; it disconnects the caller from the monitor, thus freeing the monitor for use by another process.
  - It also takes a process from the parameter

Message Passing

- Competition Synchronization with Message Passing: Example:
- a shared buffer.
- Encapsulate the buffer and its operations in a task.
  - Competition synchronization is implicit in the semantics of accept clauses.
  - Only one accept clause in a task can be active at any given time.

Java Threads

Competition Synchronization with Java Threads:

A method that includes the synchronized modifier disallows any other method from running on the object while it is in execution.

If only a part of a method must be run without interference, it can be synchronized.

Cooperation Synchronization with Java Threads:

The wait and notify methods are defined in Object, which is the root class in Java, so all objects inherit them.

The wait method must be called in a loop.

Example - the queue.

Exception Handling

In a language without exception handling:

➢When an exception occurs, control goes to the

operating system, where a message is displayed and the program is terminated. *In a language with exception handling*:

➢Programs are allowed to trap some exceptions, thereby providing the possibility of fixing the problem and continuing.

Design Issues for Exception Handling

- How and where are exception handlers specified and what is their scope?
- How is an exception occurrence bound to an exception handler?
- Where does execution continue, if at all, after an exception handler completes its execution?
- How are user-defined exceptions specified?
- Should there be default exception handlers for programs that do not provide their own?
- Can built-in exceptions be explicitly raised?
- Are hardware-detectable errors treated as exceptions that can be handled?
- Are there any built-in exceptions?
- How can exceptions be disabled, if at all?

Ada Exception Handling

Def: The frame of an exception handler in Ada is either a subprogram body, a package body, a task, or a block.

Because exception handlers are usually local to the code in which the exception can be raised, they do not have parameters.

Handler form: exception

when exception_name {| exception_name} =>

statement_sequence

…

when …

…

[when others =>statement_sequence]

- Handlers are placed at the end of the block or unit in which they occur.

Binding Exceptions to Handlers

- If the block or unit in which an exception is raised does not have a handler for that exception, the exception is propagated elsewhere to be handled.

  Procedures - propagate it to the caller.

  Blocks - propagate it to the scope in which it occurs.

  Package body - propagate it to the declaration part of the unit that declared the package (if it is a library unit (no static parent), the program is terminated).

  Task - no propagation; if it has no handler, execute it; in either case, mark it "completed".

C++ Exception Handling

- try {
- code that is expected to raise an exception} catch (formal parameter) {
- handler code
- }…..
- catch (formal parameter) {
- handler code
- }
  - catch is the name of all handlers--it is an overloaded name, so the formal parameter of each must be unique.
  - The formal parameter need not have a variable.
  - It can be simply a type name to distinguish the handler it is in from others

Java Exception Handling

The finally Clause:

Can appear at the end of a try construct Form:

```
finally {
...
}
```

Purpose: To specify code that is to be executed, regardless of what happens in the try construct.

A try construct with a finally clause can be used outside exception handling try {

```
for (index = 0; index < 100; index++) {
...
if (...) {
return;
}
```

Evaluation

The types of exceptions makes more sense than in the case of C++.

The throws clause is better than that of C++

(The throw clause in C++ says little to the programmer).

The finally clause is often useful.

The Java interpreter throws a variety of exceptions that can be handled by user programs.

Introduction to logic programming

Logic programming languages, sometimes called *declarative* programming Languages.

Express programs in a form of symbolic logic.

Use a logical inferencing process to produce results.

Declarative rather that procedural:

–Only specification of *results* are stated (not detailed *procedures* for producing them).

Proposition:

A logical statement that may or may not be true.

–Consists of objects and relationships of objects to each other.

Symbolic Logic:

Logic which can be used for the basic needs of formal logic:

–Express propositions.

–Express relationships between propositions.

–Describe how new propositions can be inferred from other propositions. (Particular form of symbolic logic used for logic programming called *predicate Calculus)*

Object Representation

Objects in propositions are represented by simple terms: either constants or variables.

Constant: a symbol that represents an object.

Variable: a symbol that can represent different objects at different times. –

Different from variables in imperative languages.

Compound Terms:

Atomic propositions consist of compound terms.

Compound term: one element of a mathematical relation, written like a mathematical function.

–Mathematical function is a mapping.

–Can be written as a table.

Parts of a Compound Term:

Compound term composed of two parts:

Example

Functor: function symbol that names the relationship.

–Ordered list of parameters (tuple).

Examples:

student(jon) like(seth, OSX) like(nick, windows) like(jim, linux)

Forms of a Proposition

Propositions can be stated in two forms:

–*Fact*: proposition is assumed to be true.

–*Query*: truth of proposition is to be determined.

Compound proposition:

–Have two or more atomic propositions.

–Propositions are connected by operators.

Clausal Form

Too many ways to state the same thing

-Use a standard form for propositions.

Clausal form:

–B1 B2 … Bn A1 A2 … Am

–means if all the As are true, then at least one B is true.

Antecedent: right side.

Consequent: left side.

Predicate Calculus and Proving Theorems

- -use of propositions is to discover new theorems that can be inferred from known axioms and theorems.
- *Resolution*: an inference principle that allows inferred propositions to be computed from given propositions r*esolution.*
- *Unification*: finding values for variables in propositions that allows matching process to succeed.
- *Instantiation*: assigning temporary values to variables to allow unification to succeed after instantiating a variable with a value,

Theorem Proving

-Basis for logic programming.

-When propositions used for resolution, only restricted form can be used.

Horn clause - can have only two forms.

–*Headed*: single atomic proposition on left side.

–*Headless*: empty left side (used to state facts).

-Most propositions can be stated as Horn clauses.

Basic Elements of Prolog

Terms:

-Edinburgh Syntax.

Term: a constant, variable, or structure.

Constant: an atom or an integer.

Atom: symbolic value of Prolog. Atom consists of either:

–a string of letters, digits, and underscores beginning with a lowercase letter.

–a string of printable ASCII characters delimited by apostrophes.

Terms: Variables and Structures

-*Variable*: any string of letters, digits, and underscores beginning with an uppercase letter.

-*Instantiation*: binding of a variable to a value.

–Lasts only as long as it takes to satisfy one complete goal.

-*Structure*: represents atomic proposition

functor(*parameter list*).

Fact Statements

-Used for the hypotheses.

-Headless Horn clauses:

female(shelley).

male(bill).

father(bill, jake).

Rule Statements

-Used for the hypotheses.

-Headed Horn clause:

Right side: *antecedent* (***if*** part)

–May be single term or conjunction.

Left side: *consequent* (***then*** part).

–Must be single term.

Conjunction: multiple terms separated by logical AND operations (implied)

Example Rules:

ancestor(mary,shelley):- mother(mary,shelley).

Can use variables (*universal objects*) to generalize meaning: parent(X,Y):-

mother(X,Y).

parent(X,Y):- father(X,Y).

grandparent(X,Z):- parent(X,Y), parent(Y,Z).

sibling(X,Y):- mother(M,X), mother(M,Y),

father(F,X), father(F,Y).

Goal Statements

-For theorem proving, theorem is in form of proposition that we want system to prove or disprove – *goal statement.*

-Same format as headless Horn **eg:** man(fred)

-Conjunctive propositions and propositions with variables also legal goals.

eg: father(X,mike)

Inferencing Process of Prolog

-Queries are called goals.

-If a goal is a compound proposition, each of the facts is a subgoal.

-To prove a goal is true, must find a chain of inference rules and/or facts.

For goal Q:

:- A

:- B

…

:- P

-Process of proving a subgoal called matching, satisfying, or resolution.

Simple Arithmetic

-Prolog supports integer variables and integer arithmetic.

-is operator: takes an arithmetic expression as right operand and variable as left operand.

eg: A is B / 17 + C

-Not the same as an assignment statement!

Example: speed(ford,100). speed(chevy,105). speed(dodge,95). speed(volvo,80).

time(ford,20).

time(chevy,21).

time(dodge,24).

time(volvo,24).

distance(X,Y) :- speed(X,Speed), time(X,Time),

Y is Speed * Time.

Trace

-Built-in structure that displays instantiations at each step.

-*Tracing model* of execution - four events:

–*Call* (beginning of attempt to satisfy goal).

–*Exit* (when a goal has been satisfied).

–*Redo* (when backtrack occurs).

–*Fail* (when goal fails).

Example

likes(jake,chocolate).

likes(jake,apricots).

likes(darcie,licorice).

likes(darcie,apricots).

trace.

likes(jake,X),

likes(darcie,X).

Bindings andscope

A PROLOG program consists of one or more relations.

The scope of every relation is the entire program.

It is not possible in PROLOG to define a relation locally to another relation, nor to group relations into packages.

Control

- In principle, the order in which resolution isdone should not affect the set of answersyielded by a query (although it will affect the order in which these answers are found).
- In practical logic programming, however, the order is very important

Deficiencies of prolog

Resolution order control

Closed word assumption: When an assertion is tested, therefore, success means true and failure means either unknown or false. As this is rather inconvenient, PROLOG bends the rules of logic by ignoring the distinction between unknown and false. In other words, an assertion is assumed to be false if it cannot be inferred to be true. This is called the ***closed world assumption***

Negation problem.

Applications of Logic Programming

Relational database management system:

RDBMS stores data in the form of tables and queries.

Prolog can replace the DML,DDL and query language which are implanted in imperative languages.

Expert Systems

Expert systems consists of database of facts, an inferencing process, a human interface to look like an expert human consultant.

Logical programming helps to solve the incompleteness of database.

Applications of logic programming(cont..)

Natural language processing

Few kinds of natural processing languages can be done using logical programming