# Chapters 11 & 12: Abstraction and Object-Oriented Programming

## Explain fundamental OOP concepts:

### Abstraction – Designing models to represent application domain entities

#### The implementation of an abstraction in an oop language

#### Programmers define ADT’s to abstract application entities

### Encapsulation – Data representation and legal operations on instances of an ADT are defined in a single syntactic unit

#### A class encapsulates its data and procedure members

### Data Hiding­ – The data representation of the abstraction is hidden from other program units

#### Interface provides methods into the abstraction

#### Enhances readability and safety by controlling how data members are accessed

### Inheritance – Software reuse through extension of an existing abstraction

#### Superclass: Extended class

#### Subclass: Extension of superclass

### Polymorphism – “Polymorphic reference” is a superclass reference variable that refers to a subclass objects

#### Polymorphic references to overridden methods cannot be resolved until run-time

## Explain and evaluate design issues for OOP

### How are ADT's encapsulated?

### How are member properties accessed?

### What visibility options are available?

### What types of inheritance are allowed?

### How is the interface of an abstraction published?

### What types of ADT operations are built-in and what operations can be defined?

### Can abstract types be parameterized?

## Analyze multiple inheritance as a language feature (advantages / disadvantages)

### Multiple Inheritance – More than one parent

### Advantages:

#### Sometimes it is natural and convenient

#### Ex: Parents ContainerSet, Iterator <- child ArrayList

### Disadvantages

#### Implementation complexity (in part due to name collisions)

#### Execution Cost – Dynamic binding is more complex and costly

## Explain storage allocation options for objects, and the significance of deep vs. shallow copy for constructors

### Can be static, static-dynamic or heap-dynamic

### If storage is mixed, assignment is complex – is data copied or a reference?

## Describe and interpret visibility modes for Java

### Public – accessible everywhere

### Protected – accessible to subclasses

### Private – accessible within class only

## Explain the significance of friend functions

### Provide private member access to unrelated classes or functions

## Explain reference polymorphism and evaluate its usage

### Polymorphic Reference- a superclass reference variable that refers to a subclass objects

#### Ex: If Beta extends Alpha, then a PR is: Alpha b = Beta();

## Explain how dynamic binding relates to polymorphism and identify when dynamic binding must occur within a given program

### Dynamic method binding: the process of determining the correct version of an overridden method to invoke

### Polymorphic references to overridden methods cannot be resolved until run-time

### If overrides are disallowed, references can be bound statically (at compile time)

#### Ex. final in Java disallows overrides

### The run-time system locates the version of the called method that is nearest to the object's actual type in the ancestral tree

## Explain the purpose of a VMT and a CIR, and create a memory map of VMTs and CIRs for a given program

### Class Instance Records (CIR)

#### Each object is represented by a class instance record

#### A CIR stores the current state (data value) of an object and provide access to methods

#### Superclass instance variables, if any, are also contained in the CIR

### Virtual Method Tables

#### Each class is represented by a VMT

#### Contains pointers to methods of the class, including inherited methods

#### Each CIR contains a reference to the VMT of its class

# Chapter 13: Concurrency

## Definitions:

### Thread of Control: a series of instructions being executed

### Thread: execution of a thread control (heavy or lightweight)

### Task: a heavyweight thread (OS level)

### Concurrency: Simultaneous execution of multiple threads of control

### Scalability: Adding processors reduces execution time

#### Scalability usually limited

### Synchronization: managing shared concurrent access to a resource

### Cooperative Synchronization: Tow threads cooperate (producer consumer)

### Competitive Synchronization: 2 threads want same resource

### Synchronous threads: threads that wait for each other

#### Blocking: waiting for another thread

### Asynchronous threads: thread that execute independently, with no waiting

#### Non-blocking: doing other work while waiting for notification from other thread

## Explain and evaluate design issues related to concurrency

### Heavyweight vs. lightweight tasks

#### Heavy Weight: Has a unique address space

#### Light Weight: Shared address space (VM)

### Alternatives for communication & synchronization for concurrent processes

#### Shared Memory

##### Threads share access to common memory area

##### Access must be synchronized

#### Message Passing

##### Message facility allows thread to send message to another thread

##### Message system required

##### Concurrent processes communicate by sending and receiving messages

##### May be on different processors

##### Advantages:

###### Simple, easy to implement

###### Flexible

###### Good for distributed systems (processes on different computers)

##### Disadvantages:

###### Low-level, no automatic synchronization

###### Expensive

#### Semaphore –

##### Access requests are queued, and then released when desired resource is free

##### Programmer is responsible for correct use of semaphore operations

##### Advantage: Effective synchronization if used correctly

##### Disadvantages:

###### Programmers must code it correctly

###### Programmer error results in deadlock

#### Monitors

##### Resource locking is encapsulated in the resource

##### Competing processes are queued to wait for lock

##### Synchronization is automatic

##### Advantages:

###### Operation in encapsulated in the resource

###### Automatic operation – less chance for error

##### Disadvantage:

###### Waiting processes are blocked

## Explain data race, deadlock, and livelock conditions, and their solutions

### Data race - 2 threads access the same resource simultaneously, interfering with each other

#### Solution: synchronization

##### Mutex: a synchronization “lock object”

### Deadlock – 2 threads wait for each other to release a desired resource

### Livelock – deadlock with unsuccessful resolution attempts

#### Solution of Both:

##### Application level resource management schemes

###### Priority, random access ects

## Interpret Java code for synchronization

# Chapter 14: Exception and Event Handling

## Definitions

### Exception: an unusual event that requires immediate processing

#### May be erroneous or not

#### Detectable by either hardware or software

### Exception handling: special processing required after detection of an exception

### Exception handler: The code unit invoked to handle an exception

## Explain and evaluate design issues for exception handling

### How are exceptions defined?

#### ADA

##### Handlers are placed at the end of a subprogram, package, task or block

#### C++

##### Uses an enclosing try/catch block

#### Java

##### Also uses try/catch with finally

### How are exceptions created?

#### They can be created by user

#### Some are already set

### How are exceptions bound to handlers?

#### ADA –

##### If the unit throwing an exception has a handler, use it. Otherwise propagate the exception

##### Exception conditions can be disabled with compiler directive

#### C++

##### The exception handler is selected based upon the type of the expression that is thrown: throw [expression];

##### Exception propagation:

###### An unhandled exception is propagated:

First enclosing try block (static)

Then to the calling procedure (Dynamic)

###### This continues until and appropriate handler is found (correct data type)

###### If no handler found, program terminated

#### Java

##### Similar to C++

###### Each handler specifies an exception type

###### An exception is bound to the first handler specifying the thrown type or its ancestor

##### Handlers can throw the same exception (after initial processing) or throw a new exception

### Where does execution resume following exception handling?

#### ADA

##### The block or unit that raises an exception but does not handle it is always terminated

##### Any block or unit to which an exception is propagated and that does not handle it is also terminated

##### Execution resumes at the unit that handles the exception

#### C++

##### After a handler completes its execution, control flows to the first statement after the last handler in the sequence of handlers of which it is an element

##### Functions can declare the exceptions that they may throw in a “throw” clause

###### Without such a specification a function can throw any exception

#### Java

##### If no handler is found in the enclosing try construct, the search is continued in the nearest enclosing try

##### If no handler is found in that method, the exception is propagated to the method’s caller

##### If no handler is found at all the program is terminated and the JVM handles the exception

##### A generic exception handler can be included in any try construct

## Describe alternatives for exception handling for languages with no formal mechanisms

### Strategies for languages that do not support exception handling:

#### Handle all exceptions at their source

##### Each program unit must be prepared to handle all possible exceptions

##### Disadvantage:

###### Limited flexibility

###### Exception handlers are numerous

#### Return an error/exception code from producers (use program logic)

##### Exceptions are indicated by a special return value

##### Disadvantage:

###### Exceptions are integrated with normal program logic, which is confusing

#### Pass a reference to an exception handler as a parameter (subprogram)

##### The exception handler is passed in

##### Disadvantage:

###### Needs to be passed through several leves

###### Mixed with program logic - confusing

#### Let the operating system handle exceptions (program termination)

##### OS handles errors, displays messages ect

##### Program terminates after exception

##### Disadvantage: no opportunity to correct error

#### Formal Exception Handling Advantages

##### Exception handling is separated from program logic

##### Exception can be propagated, allowing reuse of exception handling code

##### The compiler can generate exception checks automatically

##### Exception handling is more standardized

##### Programmers are encouraged to consider handle exceptions

## Interpret exception handling in Java code