Concepts in Prog Langs - Part 3

/

What are two important goals of modularity?

-

1) Allow one module to be written with little knowledge of the code in another module.

2) Allow a module to be redesigned and reimplemented without modifying other parts of the system.

/

How did Dijkstra’s 1969 paper, ‘Structured Programming,’ argue a program should be designed?

-

Outline the major tasks, successively refine into smaller subtasks until those tasks are expressible in basic operations.

/

What else can be refined besides tasks?

-

Data structures.

/

What is another term for the refinement into smaller tasks, often associated with recursive algorithms?

-

divide and conquer

/

Describe the development method of prototyping?

-

Implementing parts of a program in a simple way to understand if the design will really work.

/

What is a term for a meaningful part of a program, partially independent of other parts of a program? (not module)

-

Component

/

Name two important concepts in modular development.

-

1) Interface – description of visible component parts.

2) Specification – description of behavior of component, as observable through its interface.

/

What is a simple example of a program component?

-

A function

The interface (function header) is the name, parameters, and return type.

The specification describes relationship between parameters and return type.

/

Name an advantage of a system where components rely only on stated specifications?

-

Components can be replaced by any other component that satisfies the specification.

/

In programming languages, what is an abstraction mechanism?

-

Something that emphasizes the general properties of some segment of code and hides the details.

/

Two common terms associated with abstraction are client and implementation. What do they mean and how do they interact?

-

client – part of a program that uses a component

implementation – part of a program that defines a component.

Interaction between the client of an abstraction and the implementation of the abstraction is usually restricted to a specific interface.

/

What is one of the oldest abstraction mechanisms in programming languages?

-

The procedure or function (procedural abstraction)

/

Name three reasons to encapsulate code into a function.

-

1) Well-defined interface

2) Information hiding

3) Re-use

/

What are two types of data-type abstractions?

-

1) Abstract data-type declarations

2) Modules

/

What two things make up an abstract data type?

-

1) A type

2) A specified set of operations

/

What should be the only way to use an abstract data type?

-

Through its interface.

Programs should be restricted so that only the declared operations of an abstract type can be applied.

/

What is representational independence?

-

When different representations do not affect program behavior.

So a type has respresentation-independence when different implementations are indistinguishable by clients of the type.

/

What are three general groups for the operations on a type?

-

Constructors – build elements of the type.

Operators – map elements of the type to other elements of the type.

Observers – return results of some other type.

[??? – seems more applicable to low level types]

/

What is a module?

-

A programming language construct that allows a number of declarations to be grouped together.

/

In Modula, what are the two types of modules?

-

1) A definition module (interface)

2) An implementation module

/

What does type parameterization allow?

-

The ability to have multiple versions of the same abstract data type, such as stack or queue, work with different types of data, such as int or string.

(parametric polymorphism)

/

What are parameterized structures called in ML?

In C?

-

Functors

Templates

/

What makes up an object?

-

A set of operations on some hidden data.

/

What do objects provide?

-

A uniform way of encapsulating almost any combination of data and functionality.

/

How do interactions with an object occur?

-

Through simple operations called messages or member-function calls.

/

What four features make a language object oriented?

-

1) Dynamic lookup

2) Abstraction

3) Subtyping

4) Inheritance

/

What is object oriented design and how did Grady Booch suggest about going about it?

-

Identifying important concepts and using objects to structure the way these concepts are embodied in the software system.

Iteratively

1) Identify objects at a level of abstraction

2) Identify semantics (behavior) of these objects

3) Identify relationships among objects

4) Implement the objects

/

For complex task, what two things should be refined together?

-

Procedures and the data structures.

Generally easier in object oriented languages than procedural ones.

/

In common object-oriented languages such as Smalltalk, C++, and Java, what determines the implementation of an object?

What do we call a created object?

-

Its class

An instance

/

What are the function parts of an object generally called?

What about the data parts?

-

methods or member functions

instance variables, fields, or data members.

/

What is dynamic lookup?

-

The ability for the object to choose how it responds to a message. So different objects may respond to the same message in different ways. Methods are selected dynamically at runtime. (not to be confused with overloading – mechanism based on static types of operands)

/

What is abstraction?

-

Hiding implementation details in a programming unit behind an interface.

/

What is subtyping?

-

The ability to use one object in place of another if it has all the required functionality.

/

What is inheritance?

-

The ability to reuse the definition of an object to make another object.

/

What is a type of object-oriented language besides class based?

What is the difference?

What is an example language?

-

delegation-based

objects are created from other objects directly

an example is Dylan

(also JavaScript – prototypical inheritance == delegation-based?)

/

What is multiple dispatch?

Single dispatch?

-

If x is an object, then a call to a function f in x with variables y and z, becomes f(x, y, z) since the results depend on x. Multiple dispatch takes into account all three objects x, y, z in the lookup, where as single dispatch just looks at x. Multiple can be useful in things like equality testing, but tends to lose encapsulation, and is not the norm (not in Java, Smalltalk, C++)

/

What is the primary advantage of subtyping?

-

It allows uniform operations over various types of data.

/

What code smell does inheritance help prevent?

-

Code duplication

/

With inheritance, what are the three views of a class?

What are these interfaces called?

-

1) Implementation view – private interface

2) Client view – public interface

3) Inheritance view – protected interface

/

What is a way to simulate objects (the dynamic lookup portion) in non-oop languages with higher order functions?

What is still missing?

-

With closures – put the private data in the static scope.

Subtyping and inheritance will still be missing.

/

What is the difference between subtyping and inheritance?

-

Subtyping is a relationship on interfaces.

Inheritance is a relationship on implementation.

/

What is the main difference in the structural organization of function-oriented languages versus object-oriented ones?

What does this make easier and harder in each?

-

In function-oriented, data is organized separately from functions, in object-oriented data and functions are grouped together.

So in function-oriented programs it is easy to add new operations but difficult to add new types of data, whereas in object-oriented programs it is easy to add new types of data, but difficult to add new operations.

/

What is a design pattern?

-

A general solution that has come from the repeated addressing of similar problems.

/

What was the first language with objects?

What language refined them?

-

Simula

Smalltalk (everything is an object)

/

What do advocates of objects-oriented languages mean when they use the term “polymorphism?”

-

dynamic lookup (subtype polymorphism) – the same message name invokes different code depending upon the object that receives the message.

/

In Smalltalk, how do you reference the object receiving the message?

-

self

/

In Smalltalk, how do you search for a message in a parent class?

-

super

/

Is subclassing different from subtyping in Smalltalk?

-

Yes, a subclass may not implement all the methods of its superclass. Also, two classes could be defined independently and have the same functionality, so they are the same types but there is no inheritance.

So, subtyping is a relationship between types, a property important to users, while subclassing is about inheritance, a property of implementations.