

# Reusable Software

## Lecture Plan:

- What is Reusable Software?
- Designing for Reuse
- Libraries

# What is Reusable Software?

All software is constructed from components

These components may be:

- classes (object-oriented programming style)
- modules (modular programming style)
- routines (early programming style)

Reusable software is any software component that can be used in more than one program/system

Most components are potentially reusable, but only some are really worth reusing

## Definition: reusability

Reusability is the ability of software elements to serve for the construction of many different applications.

*Object-Oriented Software Construction*  
Bertrand Meyer

# Reusability

Reusability has an influence on all other aspects of software quality, for solving the reusability problem essentially means that less software must be written, and hence that more effort may be devoted (for the same total cost) to improving the other factors, such as correctness and robustness.

*Object-Oriented Software Construction*  
Bertrand Meyer

## Obstacles to Reuse (Psychological)

- “I can write a better one myself.”
- “I don’t trust other people’s code.”
- “It will take me longer to understand this stuff than to write my own.”
- “I don’t need anything this general.”

## Obstacles to Reuse (Real)

### Technical:

- Some components really are badly designed or written.
- Some interfaces really are unnecessarily hard to use.
- Documentation is frequently inadequate.

### Commercial:

- Source code kept secret.
- Price too high.
- Unacceptable licence conditions.
- Inadequate service (e.g. bugs not fixed promptly).

## Good Reusable Components

- implement a useful concept or service
- have simple, easy-to-use interfaces
- are adaptable and extendable (via generic classes and inheritance)
- are properly documented and tested
- meet code quality standards
- are adequately supported and maintained

# Designing for Reuse

- A class should implement a single concept.

✓ stack of things

✗ stack of things with print command

- A class should implement a concept in its most general form.

✓ stack of things

✗ stack of numbers

✓ sort things into specified order

✗ sort numbers into ascending order

- A class should implement a complete concept.

✓ matrix arithmetic

✗ matrix add, subtract & multiply

- Separate the concept from its implementation.

✓ `table.lookup(key)`

✗ `table.hash_search(hash_code)`

# Reusability and COMP1120

The design and implementation of reusable software draws on many of the topics discussed in this course, including

- software design principles
- abstract data types
- the inheritance mechanism
- generic classes
- deferred classes

# Libraries

**Basic libraries** Reusable classes should be available to cover the most frequently needed data structures and algorithms.

**Library evolution mechanisms** Mechanisms should be available to facilitate library evolution with minimal disruption of client software.

*Object-Oriented Software Construction*

Bertrand Meyer

Libraries are the main repositories of reusable code, and are the main mechanism for software reuse

# Libraries Contain

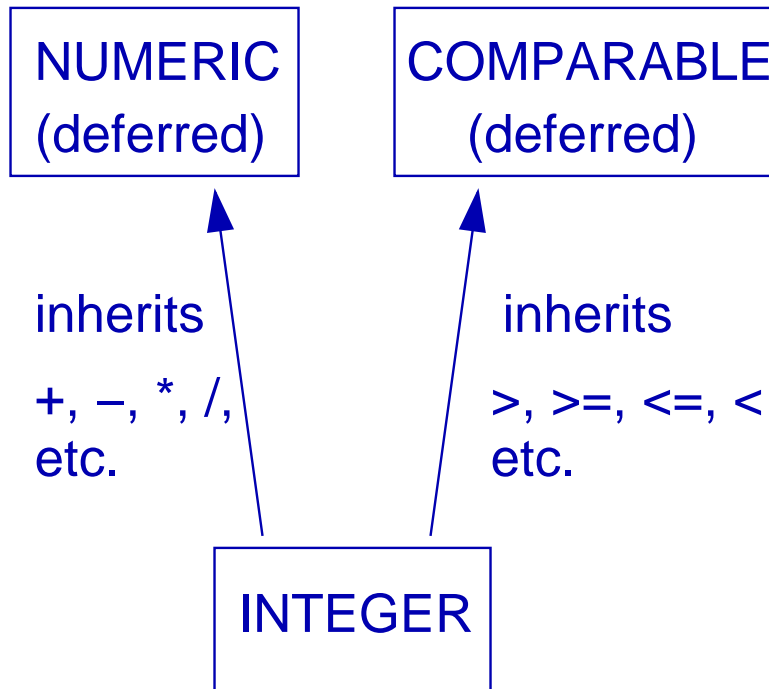
- implementations of components (classes/modules/routines)
- documentation for users
  - what the components do
  - how to use them
  - interface details
- support for software maintenance (usually kept private)
  - documentation (source code, design docs, etc.)
  - test routines

# Understanding Class Libraries

Most reusable software is contained in libraries. If you want to reuse existing classes, or create your own, then you need to understand how class libraries are designed and organized. This takes some effort because:

1. Class libraries tend to define large, elaborate class hierarchies containing many instances of multiple inheritance, deferred classes and generic classes.
2. Library classes tend to have a great many features, including large numbers of inherited features.

## From the GNU Eiffel Library

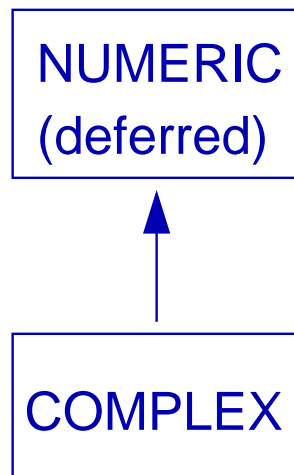


- *NUMERIC* describes the general properties of numbers.
- *COMPARABLE* describes things that possess a strict order.
- *INTEGER* describes the set of integers, which are objects that are both numbers and things that can be ordered.

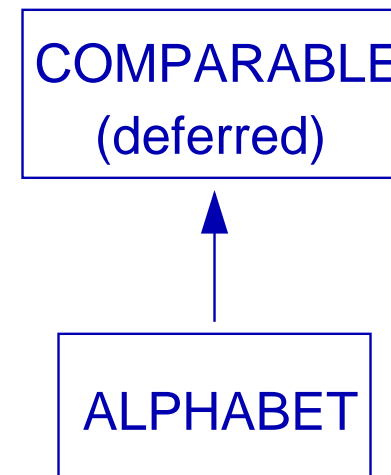
## From the GNU Eiffel Library

This arrangement separates the concept of 'number' from that of 'order', so that the user can create:-

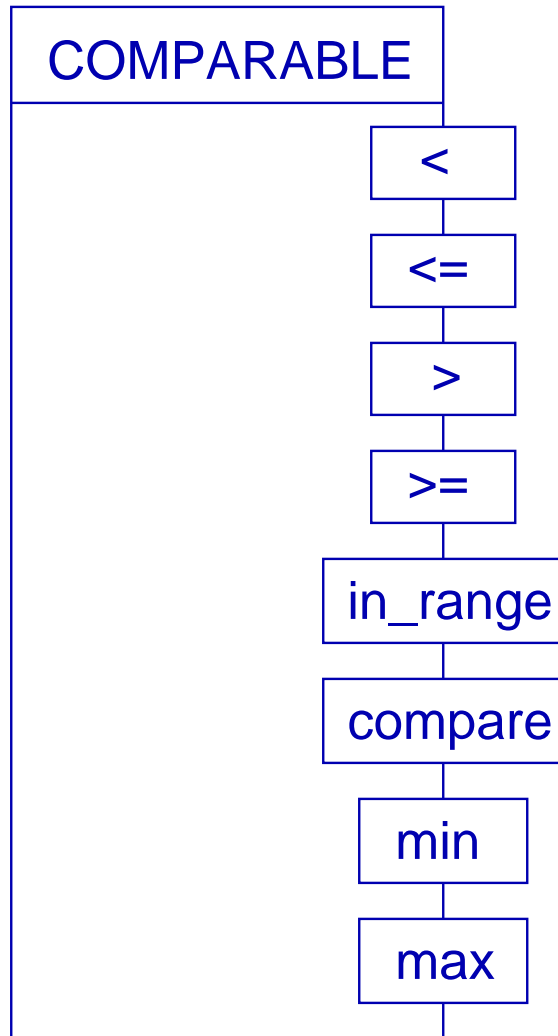
- things that are numbers but are not ordered



- things that are ordered but are not numbers



## A Closer Look at *COMPARABLE*



This class defines a *carefully chosen* set of features to embody the concept of ordering. Observe that

- Most features are there for convenience rather than necessity.
- Every feature involves the concept of ordering.
- The equality operators `=` and `/=` have been omitted. (Equality and order are different concepts.)

Most library classes have been carefully designed, and are worth studying.

## Suggested Reading

From the Textbook:

Thomas, Peter & Weedon, Ray (1997) *Object-Oriented Programming in Eiffel*, 2nd Edition, Addison-Wesley, England

Chapter 3 “Abstract Data Types” pp 67–82

Chapter 15 “Client-Supplier vs Inheritance” pp 423–431

From the COMP1110 Reading Brick:

- **Understanding Object-Oriented: A Unifying Paradigm**

Korson, Tim & McGregor, John D “Understanding Object-Oriented: A Unifying Paradigm” *Communications of the ACM*, Vol 33 No 9 September 1990, pp 40–60,