

5.5 Code Reuse

- Idea: Don't invent the wheel twice
 - Reuse code written by others
- Process
 - Search for reusable code
 - Check whether it's what you need
 - Check whether it works
 - Learn how to use it
 - Integrate it into your code
- Need to be flexible
 - adapt requirements to what's out there
 - adapt design
 - often cheaper to change your mind about requirements than to redevelop code you could have reused

Code Reuse

- Advantages
 - saved effort
 - higher quality
 - many open source libraries are much better than anything you could write
 - standardization
 - other people may be using the same library
- Kinds of reusable code
 - Libraries
 - Frameworks
 - Components
 - Operating System Functions
 - Open Source Code

Reusable Code: Libraries

- Class or procedure libraries
- Well-documented set of related classes
- "Base" library
 - Java, C++ standard libraries
 - contain basic classes: String, Collection...
 - and sometimes much more
 - Java: Gui, networking,...
- Special purpose libraries
 - Algorithm libraries
 - Network protocols, database access, Gui, ...
- Problem
 - Often hard to combine several libraries
 - conflicting assumptions

C++ Standard Library

- ISO 14882
- formerly: Standard Template Library (STL)
- Focus on:
 - containers
 - math
 - algorithms
- Goal:
 - no performance overhead
 - as a result:
 - hard to use
 - but very powerful

So What's a Template?

- Using types as parameters
- Example: Stack in C++

```
template< class T >
class stack {
public:
    stack();           // constructor
    T& pop();
    void push( T );
    // ...
};
```

- In Java: Generics
 - introduced only in Java 1.5
 - List< T >

Java Library: Overview

- java.awt
 - platform Guis
- java.io
 - file input and output
- java.lang
 - fundamental classes, such as Object, String, Number
- java.net
 - networking, URLs, ...
- java.rmi
 - Remote Method Invocation
 - for communicating between processes (or hosts)
- java.security
- java.sql
 - Java Database Connectivity
 - accessing relational databases with SQL
- java.util
 - collections
 - event support
 - time
 - random numbers
- java.util.zip
 - compressed files in zip format
- javax.swing
 - platform-independent Gui
- many other packages

Reusable Code: Frameworks

- Similar to libraries
 - sets of closely related classes
 - frameworks define overall control-flow
 - libraries don't
- Extension points
 - Where users can "plug in" their own classes
 - through inheritance
 - through implementation of empty method bodies
 - through configuration parameters
- Examples
 - Java Swing (kind of...)
 - Component frameworks
 - runtime environments for components

Reusable Code: Components

- Idea: components can easily be plugged together without having to change them
 - components define what they provide and what they require
 - connectors
 - self-description
- Component standards
 - Dot-Net
 - COM
 - Enterprise Java Beans
 - Java Beans
- A component standard includes
 - a framework (including library)
 - standard properties of components

Reusable Code: Operating Systems

- OSes often include libraries
- MS Windows
 - Win32, MFC
 - Dot-Net
- Advantages of OS functions
 - fast
 - reliable
- Disadvantages of OS functions
 - may be hard to use
 - platform-dependent

Reusable Code: Source Code

- Advantages
 - platform-independent
 - can be adapted
- Disadvantages
 - compiling often difficult
 - different language and compiler versions
 - hard to understand
 - may be incomplete

What to look for in Reusable Code?

- Requirements
 - does the right thing
- Architecture
 - does it in a way that fits your system
 - architectures are often incompatible
 - central vs. distributed
 - functional vs. imperative
 - event-based vs. procedure-call based
 - naming conflicts
 - classes with same names
- Not buggy
- Understandability
 - how difficult to learn?
- Adaptability
 - inheritance, configuration parameters...

Product Families

- Another purpose of reuse
 - not just reusing somebody else's effort
 - also: saving space and avoiding duplicate code
- Product family
 - set of related products
 - products have common (reused) parts
 - products often developed at the same time
 - examples:
 - Mac / MS / Unix versions
 - desktop / handheld versions
 - different branches of an organization
 - embedded systems: one version per hardware product
- Solution
 - design reusable components / subsystems
 - products are different combinations of components

Possible Problems of Reusable Code

- Hard to understand
- Lack of documentation
 - unspecified dependencies on other code
- Quality is unknown
 - how well tested?
 - software quality cannot be measured objectively
- Incompatible architecture
- Incompatible language
- Difficult to adapt
 - only a small change is needed — but how to do it?
- “Not invented here”
 - irrational distrust of outside code