Programming Style Principles, (Coupling, Cohesion, Completeness) Software Development

CSc3350

Dr. William Greg Johnson
Department of Computer Science
Georgia State University

Georgia State University | COLLEGE OF ARTS & SCIENCES

What Makes High-Quality Programming Code?

- Correctness
- Readability
- Maintainability
- Documented
- Well organized (formatted)

Georgia State University | COLLEGE OF ARTS & SCIENCES

What Makes High-Quality Programming Code?

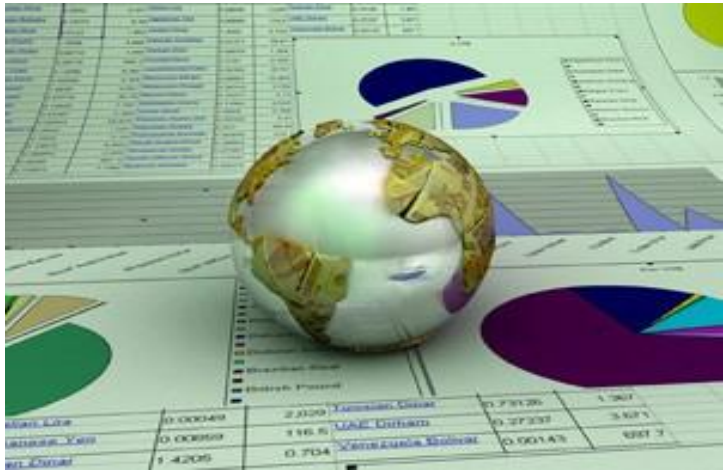
- **Good Programming Style Principles!**

Georgia State University | COLLEGE OF ARTS & SCIENCES

Content to be Covered

- Why Quality Is Important?
- Software Quality: External and Internal
- What is High-Quality Code?
- Code Conventions
- Managing Complexity
- Characteristics of Quality Code
- Coupling and Cohesion
- Packages

What is High-Quality Programming Code?



Georgia State University | COLLEGE OF ARTS & SCIENCES

Why Quality Is Important?

```
static void Main()
{
  int value=010, i=5, w;
  switch(value){case 10:w=5;System.out.println
(w);break;case 9:i=0;break;
          case 8:System.out.prinln("8");break;
     default:System.out.println("def "); for (int k
= 0; k < i; k++) System.out.println(k -
value));break;} {System.out.println("loop!"); }
}
```

What does this code do? Is it correct?

Why Quality Is Important? (2)

```
static void Main()
{
  int value = 10, i = 5, w;
  switch (value)
  {
    case 10: w = 5;
             System.out.println(w);
             break;
    case 9: i = 0;
            break;
    case 8: System.out.println(” 8");
            break;
    default:
      System.out.println("def ");
      for (int k = 0; k < i; k++)
       System.out.prinln(k+value));
      break;
    }
    System.out.println("loop!");
}
```

Now the code is formatted but is still unclear.

Software Quality

External quality

- Does the software behave correctly?
- Is the produced result, correct?
- Does the software run fast?
- Is the software UI easy-to-use?
- Is the code secure enough?

Internal quality

- Is the code easy to read and understand?
- Is the code well structured?
- Is the code easy to modify?

What is High-Quality Programming Code?

High-quality programming code:

- Easy to read and understand
  - Easy to modify and maintain
- Correct behavior in all cases
  - Well tested
- Well architectured and designed
- Well documented
  - Self-documenting code
- Well formatted

What is High-Quality Programming Code? (2)

High-quality programming code:

- Strong cohesion at all levels: modules, classes, methods, etc.
  - Single unit is responsible for single task
- Loose coupling between modules, classes, methods, etc.
  - Units are independent one of another
- Good formatting
- Good names for classes, methods, variables, etc.
- Self-documenting code style

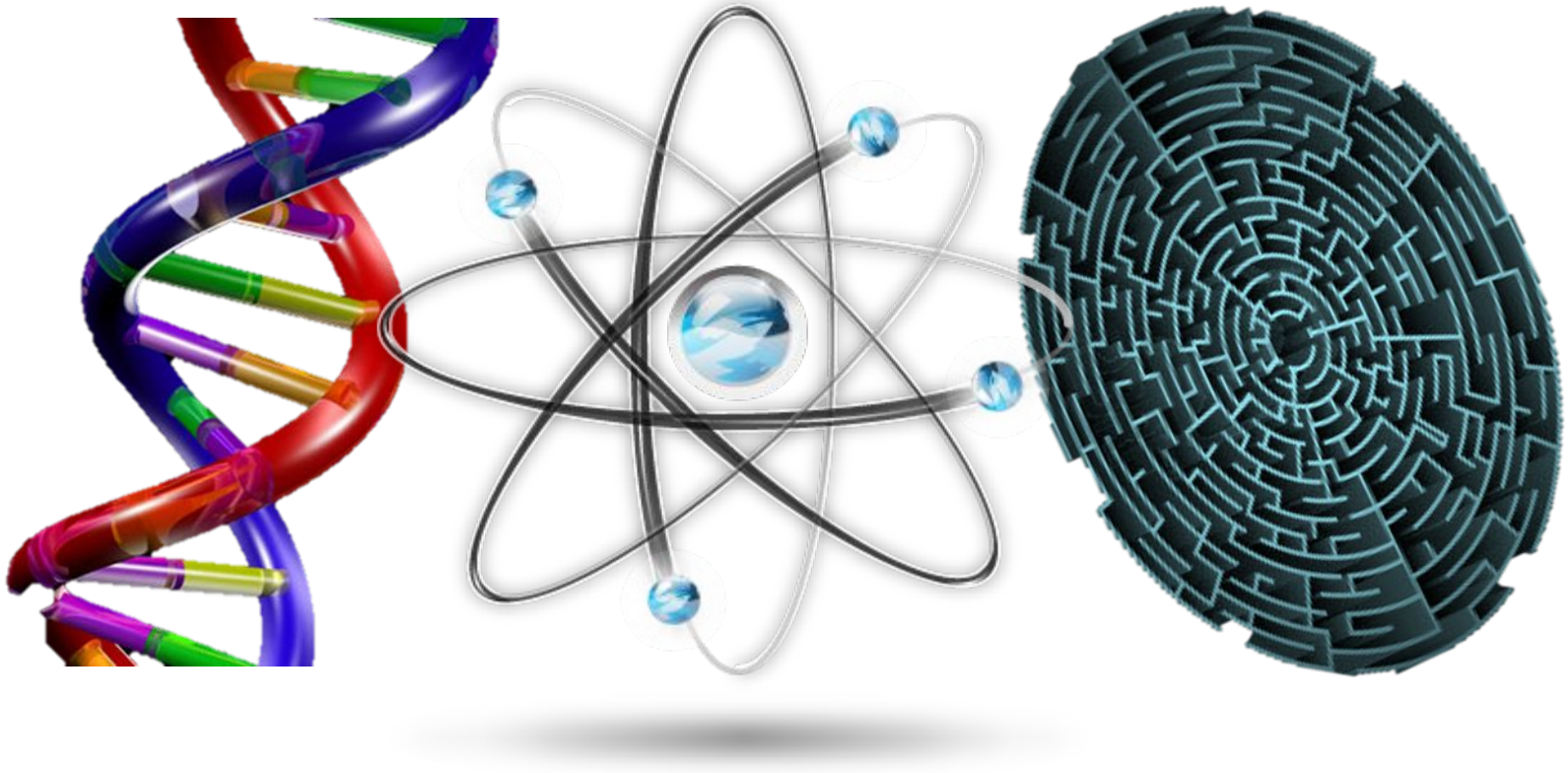
Code Conventions

- **Standards!**

Georgia State University | COLLEGE OF ARTS & SCIENCES

Code Conventions

- Code conventions are formal guidelines about the style of the source code:
  - Code formatting conventions
    - Indentation, whitespace, etc.
  - Naming conventions
    - `PascalCase` or `camelCase`, prefixes, suffixes, etc.
  - Best practices
    - Classes, interfaces, enumerations, structures, inheritance, exceptions, properties, events, constructors, fields, operators, etc.



Managing Complexity

Georgia State University | COLLEGE OF ARTS & SCIENCES

Managing Complexity

Managing complexity has central role in software construction

- Minimize the amount of complexity that anyone's brain must deal with at certain time

Architecture and design challenges

- Design modules and classes to reduce complexity

Code construction challenges

- Apply good software construction practices: classes, methods, variables, naming, statements, exception handling, formatting, comments, etc.

Managing Complexity (2)

Key to being an effective programmer:

- Maximizing the portion of a program that you can safely ignore
  - While working on any one section of code
- Most practices in software development have ways to achieve this important goal

Key Characteristics of High-Quality Code

Correct behavior

- Conforming to the requirements
- Stable, no hangs, no crashes
- Bug free – works as expected
- Correct response to incorrect usage (exceptions)
- Readable – easy to read
- Understandable – self-documenting
- Maintainable – easy to modify when required

Key Characteristics of High-Quality Code (2)

Good identifier names

- Good names for variables, constants, methods, parameters, classes, structures, fields, properties, interfaces, structures, enumerations, namespaces,

High-quality classes, interfaces and class hierarchies

- Good abstraction and encapsulation
- Simplicity, reusability, minimal complexity
- Strong cohesion, weak/small coupling

Key Characteristics of High-Quality Code (3)

High-quality methods

- Reduced complexity, improved readability
- Good method and parameter names
- Strong cohesion, weak/small coupling

Variables, data, expressions and constants

- Minimal variable scope, span, live time
- Simple expressions
- Correctly used constants
- Correctly organized data

Key Characteristics of High-Quality Code (4)

Correctly used control structures

- Simple statements
- Simple conditional statements and simple conditions
- Well organized loops without deep nesting

Good code formatting

- Reflecting the logical structure of the program
- Good formatting of classes, methods, blocks, whitespace, long lines, alignment, etc.

Key Characteristics of High-Quality Code (5)

High-quality documentation and comments

- Effective comments
- Self-documenting code

Defensive programming and exceptions

- Ubiquitous use of defensive programming
- Well organized exception handling

Code tuning and optimization

- Quality code instead of good performance
- Code performance when required

Key Characteristics of High-Quality Code (6)

Following the corporate code conventions

- Formatting and style, naming, etc.
- Domain-specific best practices

Well tested and reviewed

- Testable code
- Well designed unit tests
- Tests for all scenarios
- High code coverage
- Passed code reviews and inspections

Desired Class/Object Interaction

Maximize internal interaction (cohesion)

- easier to understand
- easier to test

Minimize external interaction (coupling)

- can be used independently
- easier to test
- easier to replace
- easier to understand

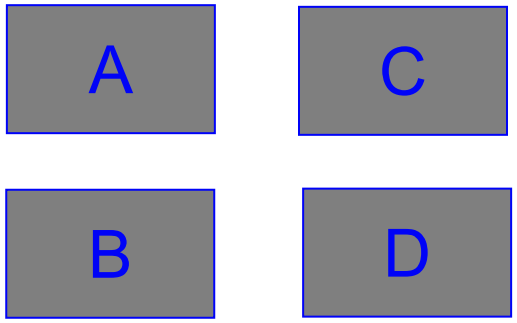
Georgia State University | COLLEGE OF ARTS & SCIENCES

Characteristics of Good Design

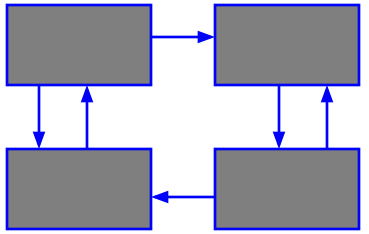
- Component independence
  - High cohesion
  - Low coupling
- Exception identification and handling
- Fault prevention and fault tolerance

Georgia State University | COLLEGE OF ARTS & SCIENCES

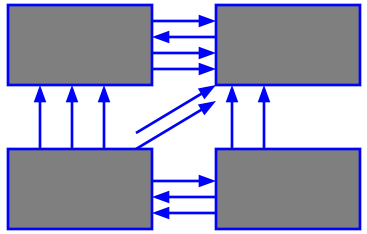
Coupling: Degree of dependence among components



No dependencies



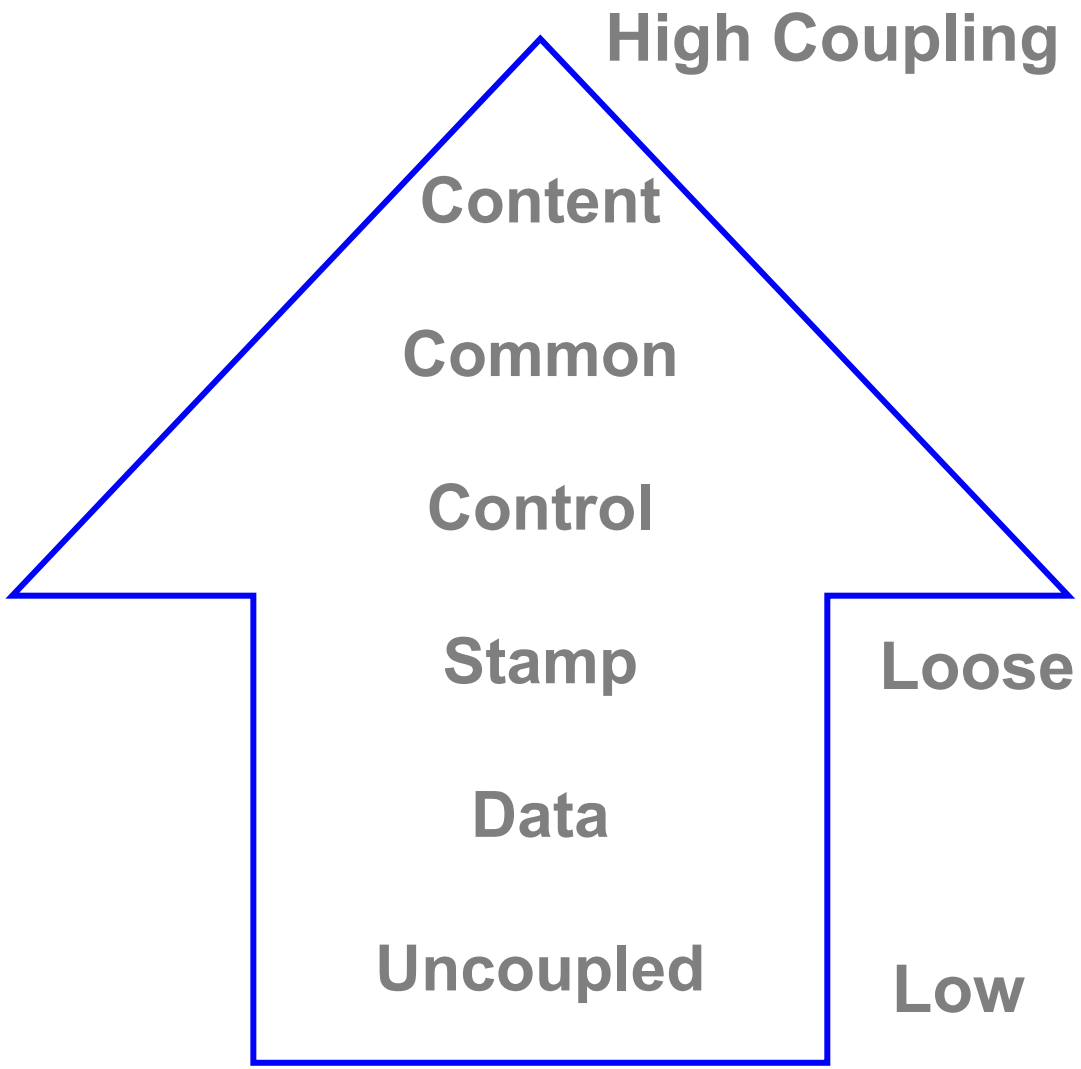
Loosely coupled-some dependencies



Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Georgia State University | COLLEGE OF ARTS & SCIENCES



Georgia State University | COLLEGE OF ARTS & SCIENCES

Content Coupling

Definition: A component directly references the content of another component

- Component ***p*** modifies a statement of module ***q***
- Component ***p*** refers to local data of module ***q*** (in terms of a numerical displacement)
- Component ***p*** branches to a local label of module ***q***

Georgia State University | COLLEGE OF ARTS & SCIENCES

Content Coupling (cont)

Content coupled components are inextricably interlinked.

- Change to component ***p*** requires a change to component ***q*** (including recompilation)
- Reusing component ***p*** requires using component ***q*** also

Typically, only possible in assembly languages

Georgia State University | COLLEGE OF ARTS & SCIENCES

Common Coupling

- Using global variables (i.e., *global coupling*)
- All components have read/write access to a global data block
- Components exchange data using the global data block (instead of arguments)
- Single component with write access where all other component have read access is not common coupling

Georgia State University | COLLEGE OF ARTS & SCIENCES

Common Coupling (cont)

- Look at many components to determine the current state of a variable
- Side effects require looking at all the code in a function to see if there are any global effects
- Changes in one component to the declaration requires changes in all others
- Identical list of global variables must be declared for components to be reused
- Component is exposed to more data than is needed

Georgia State University | COLLEGE OF ARTS & SCIENCES

Control Coupling

Definition: Component passes control parameters to coupled components.

- May be either good or bad, depending on situation.
  - Bad when component must be aware of internal structure and logic of another module
  - Good if parameters allow factoring and reuse of functionality

Georgia State University | COLLEGE OF ARTS & SCIENCES

- Acceptable: Component ***p*** calls component ***q*** and ***q*** returns a flag that indicates an error (if any)
- Not Acceptable: Component ***p*** calls component ***q*** and ***q*** returns a flag back to ***p*** that says it must output the error "I goofed up"

Georgia State University | COLLEGE OF ARTS & SCIENCES

Stamp Coupling

Definition: Component passes a data structure to another component that does not have access to the entire structure.

- Requires second component to know how to manipulate the data structure (e.g., needs to know about implementation)
- May be necessary due to efficiency factors: this is a choice made by insightful designer, not lazy programmer.

Georgia State University | COLLEGE OF ARTS & SCIENCES

Example

The printing component of the customer billing system accepts a customer data structure as an argument, parses it, and prints the name, address, and billing information.

double printEmployee(Employee _emp);

Better

```
double printEmployee(
        String Name,
        String Address,
        float billAmount);
```

Georgia State University | COLLEGE OF ARTS & SCIENCES

Data Coupling

Definition: Two components are data coupled if there are homogeneous data items.

- Every argument is simple argument or data structure in which all elements are used
- Good, if it can be achieved.
- Easy to write contracts for this and modify component independently.

Georgia State University | COLLEGE OF ARTS & SCIENCES

Primary programming design in OOP

- Object-oriented designs tend to have low coupling.

Georgia State University | COLLEGE OF ARTS & SCIENCES

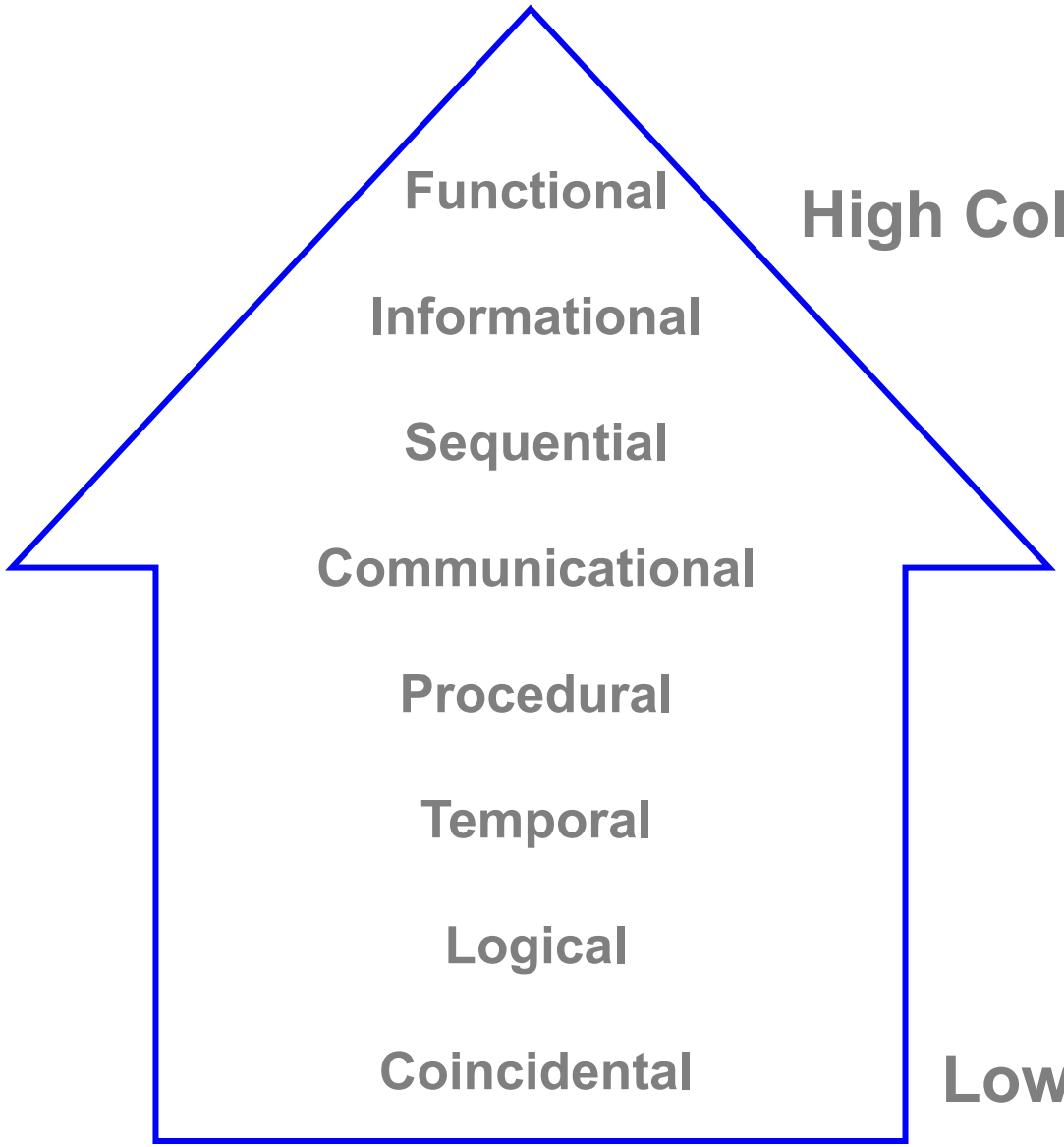
Cohesion

Definition: The degree to which all elements of a component are directed towards a single task and all elements directed towards that task are contained in a single component.

- Internal glue with which component is constructed
- All elements of a component are directed toward and essential for performing the same task
- High is good

Georgia State University | COLLEGE OF ARTS & SCIENCES

Range of Cohesion



Georgia State University | COLLEGE OF ARTS & SCIENCES

An Ordinal Cohesion Scale

6 - Functional cohesion

module performs a single well-defined function

5 - Sequential cohesion

>1 function, but they occur in an order prescribed by the specification

4 - Communication cohesion

>1 function, but on the same data (not a single data structure or class)

3 - Procedural cohesion

multiple functions that are procedurally related

2 - Temporal cohesion

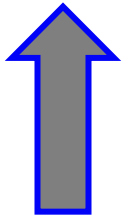
>1 function, but must occur within the same time span (e.g., initialization)

1 - Logical cohesion

module performs a series of similar functions, e.g., Java class **java.lang.Math**

0 - Coincidental cohesion

Strong cohesion



Weak cohesion

Georgia State University | COLLEGE OF ARTS & SCIENCES

Measuring Module Cohesion

- Cohesion or component "strength" refers to the notion of a module level "togetherness" seen in system abstraction levels
- Internal Cohesion or Syntactic Cohesion
  - closely related to the way in which large programs are modularized
  - ADVANTAGE: cohesion computation can be automated
- External Cohesion or Semantic Cohesion
  - externally discernable concept that assesses whether the abstraction represented by the module (class in object-oriented approach) can be a "whole" semantically
  - ADVANTAGE: more meaningful

Georgia State University | COLLEGE OF ARTS & SCIENCES

Weak Cohesion Indicates Poor Design

- Unrelated responsibilities/functions imply that a component will have *unrelated reasons to change* in the future
- Because *semantic* cohesion is difficult to automate, and automation is key, most cohesion metrics focus on *syntactic* cohesion

Georgia State University | COLLEGE OF ARTS & SCIENCES

Structural Class Cohesion

SCC measures how well class responsibilities are related

- Class responsibilities are expressed as its operations/methods

Cohesive interactions of class operations:
How operations can be related:

- Operations calling other operations (of this class)
- Operations sharing attributes
- Operations having similar signatures (e.g., similar data types of parameters)

Georgia State University | COLLEGE OF ARTS & SCIENCES

Interface-based Cohesion Metrics

Advantages

- Can be calculated early in the design stage

Disadvantages

- Relatively weak cohesion metric:
  - Without source code, it is unknow what exactly a method is doing (e.g., it may be using class attributes, or calling other methods on its class)
  - Number of different classes with distinct method-attribute pairs is generally larger than the number of classes with distinct method-parameter-type, because the number of attributes in classes tends to be larger than the number of distinct parameter types

Georgia State University | COLLEGE OF ARTS & SCIENCES

Packages and Organized Programming

- A Java package is a namespace that groups related classes and interfaces together.
- They help to organize code and make it easier to find and use.
- They also provide access control, allowing you to restrict which classes and interfaces can be accessed from the outside.

Examples:

```
import java.lang;
import java.util.ArrayList;
import java.sql.*;
```

Java Foundation Packages

Java gives many classes grouped into different packages based on functionality.

The six foundation Java packages are:

1) java.lang

   Contains classes for primitive types, strings, math functions, threads, and exception handling

2) java.util

   Contains classes such as vectors, hash tables, date etc.

3) java.io

   Stream classes for I/O

4) java.awt

   Classes for implementing GUI – windows, buttons, menus etc.

5) java.net

   Classes for networking

6) java.applet

   Classes for creating and implementing applets

Purpose of Packages

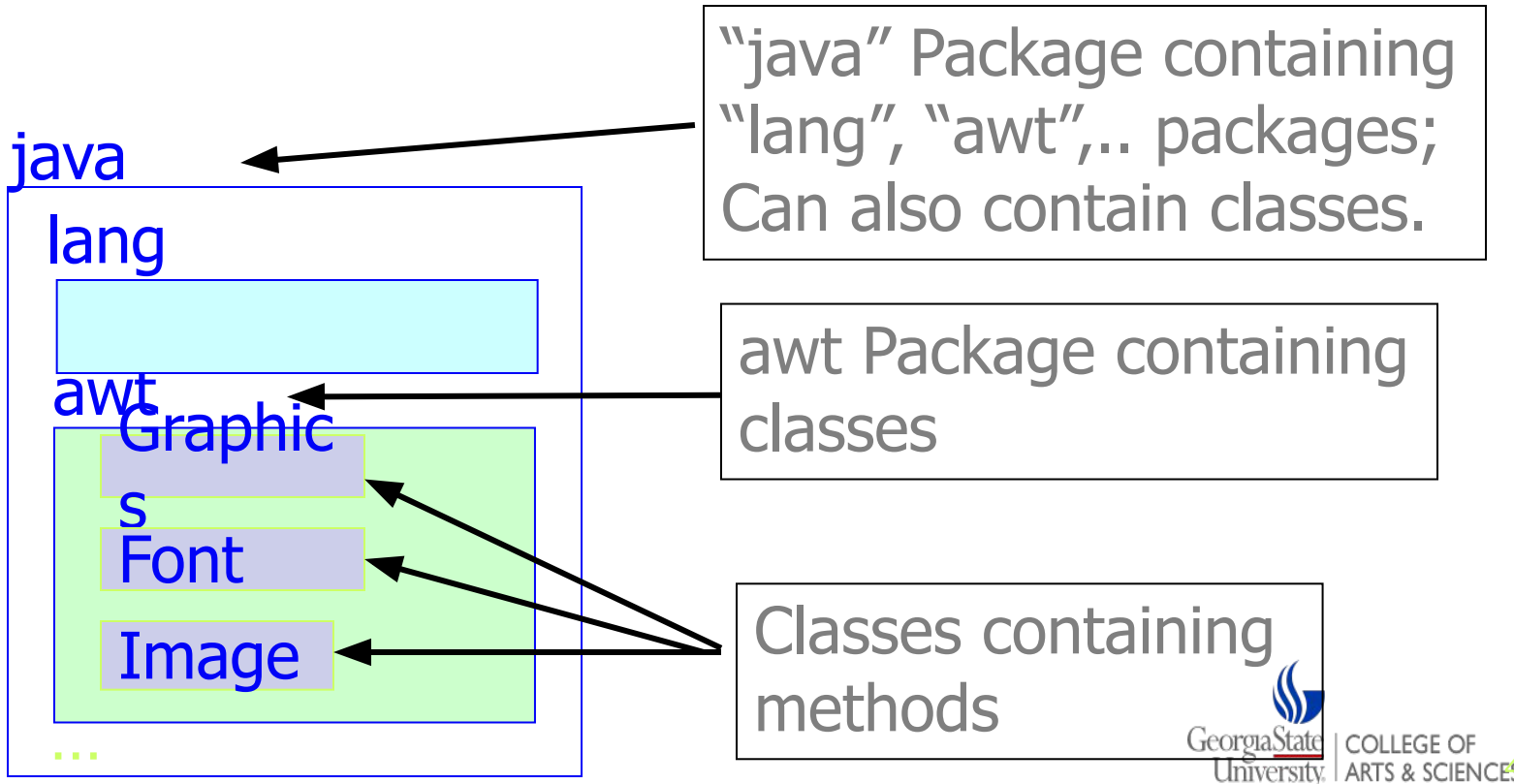
- To organize your code into logical groups. This makes your code easier to read, maintain, and understand.
- To prevent naming conflicts: For example, there could be two classes named **`employee`** in different packages.
- To control access to your code allowing you to restrict which classes and interfaces can be accessed from outside of the package. This helps to protect your code from unauthorized access.

Packages: Best Practices

- Group related classes and interfaces together. The classes and interfaces related to user accounts can be put in a package called **`com.employee.user`**
- Use descriptive package names to make your code easier to read and understand
- Use access control to protect your code from unauthorized access. Make private classes and interfaces that are only used by other classes within the package (data hiding and 'black-box' model)
- Use sub-packages to organize your code into even smaller groups. Create a sub-package called **`com.employee.user.hr`** for classes and interfaces related to user human resources.

Using System Packages

- The packages are organised in a hierarchical structure. For example, a package named "java" contains the package "awt", which in turn contains various classes required for implementing GUI (graphical user interface).



Creating Packages

- Java supports a keyword called "package" for creating user-defined packages. The package statement must be the first statement in a Java source file (except comments and white spaces) followed by one or more classes.

```
package mypackage;
public class ClassA {
     // class body
}
class ClassB {
 // class body
}
```

- Package name is "mypackage" and classes are considered as part of this package; The code is saved in a file called "ClassA.java" and located in a directory called "mypackage".

Creating Sub Packages

- Classes in one or more source files can be part of the same packages.
- As packages in Java are organised hierarchically, sub-packages can be created as follows:
  - package mypackage.Math
  - package mypackage.secondpackage.thirdpackage
- Store “thirdpackage” in a subdirectory named “mypackage\secondpackage”. Store “secondpackage” and “Math” class in a subdirectory “mypackage”.

Accessing a Package

- As indicated earlier, classes in packages can be accessed using a fully qualified name or using a short-cut if we import a corresponding package.
- The general form of importing package is:
  - import package1[.package2][…].classname
  - Example:
    - import mypackage.ClassA;
  - All classes/packages from higher-level package can be imported as follows:
    - import mypackage.*;
    - import mypackage.secondpackage.*;

Creating/storing a Package

- You store the code listing below in a file named 'ClassA.java' within subdirectory named 'mypackage' within the current directory (say 'Utilities').

```
package mypackage;
public class ClassA
{
   public void display()
  {
    System.out.println("Hello, I am ClassA");
  }
}
class ClassB
{
 // class body
}
```

Using a Package

- Within the current directory 'Utilities', store the following code in a file named 'UseClassA.java'

```
import mypackage.ClassA;
public class UseClassA
{
  public static void main(String args[])
  {
      ClassA objA = new ClassA();
      objA.display();
  }
}
```

Compiling and Running

- When 'UseClassA.java' is compiled, it compiles and places '.class' file in the current directly.
- If the '.class' file of ClassA in subdirectory "mypackage" is not found, it compiles 'ClassA.java' also.
- Note: It does not include code of 'ClassA' into 'UseClassA'
- When the program 'UseClassA' is run, java looks for 'ClassA.class' file in a package called "mypackage" and loads into memory for excution.

Attributions

1. https://www.scribd.com/presentation/108868007/Design-Principles
2. Pfleeger, S., "Software Engineering Theory and Practice", Prentice Hall, 2001
3. Chawla, Jagnesh. "Cohesion & Coupling"
4. https://www.scribd.com/presentation/604547332/0-What-is-High-Quality-Programming-Code
5. https://www.scribd.com/presentation/6426854/JAVA-TM-Package