INT103
Advanced Programming

2022/2

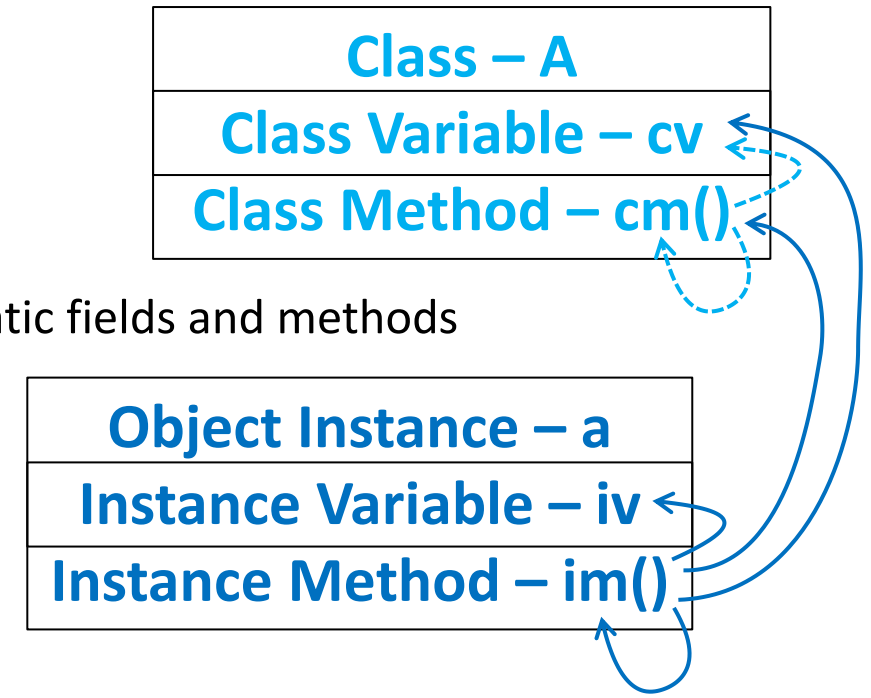
Bachelor Science in Information Technology (B.Sc.IT)

School of Information Technology (SIT)

King Mongkut's University of Technology Thonburi (KMUTT)

Object-Oriented Programming Principles

- **Class-based OOP (vs. Prototype-based OOP)**
  - **Static Fields/Attributes** (aka: **Class Variables**)
    - are shared among all objects in the class
  - **Non-static Fields/Attributes** (aka: **Instance Variables**)
    - are not shared among objects in the class
  - **Static Methods** (aka: **Class Methods**)
    - can access static fields and methods but cannot access non-static fields and methods
  - **Non-static Methods** (aka: **Instance Methods**)
    - can access both static and non-static fields and methods
- **Information Hiding / Encapsulation**
  - Private – Package-Private (default) – Protected – Public
- **Inheritance**
  - Single Inheritance vs. Multiple Inheritance
  - Interface Inheritance vs. Implementation Inheritance
- **Polymorphism**
  - Overloading (different formal parameters: different numbers of parameters, different parameter types)
  - Overriding (subclass redefines superclass's implementation)
  - Parametric Polymorphism (i.e., generics)



- **Information Hiding / Encapsulation**
  - **Private**
    - Accessible by objects in the same class
  - **Package-Private** (default)
    - Accessible by objects/classes in the same package
  - **Protected**
    - Accessible by objects/classes in the same package, and
    - Accessible by objects of their subclasses
  - **Public**
    - Accessible by all

- **Information Hiding / Encapsulation**
  - **Class**
    - **Public** – accessible by all
    - **Package-Private** (default) – accessible by others in the same package only
  - **Member** (Field/Method/Nested Class)
    - **Public** – accessible by all
    - **Protected** – accessible by others in the same package and by its subclasses
    - **Package-Private** (default) – accessible by others in the same package only
    - **Private** – accessible by others in the same class only
- **Class (Static Member) vs. Instance (Non-Static Member)**
  - static members cannot access non-static members
  - Non-static members can access static members

- **Single Inheritance vs. Multiple Inheritance**
  - **Single Inheritance**
    - There is **only one parent** to inherit from : **classes** in Java, **classes** in Smalltalk, …
  - **Multiple Inheritance**
    - There can be **multiple parents** to inherit from : **classes** in C++, **interfaces** in Java

- **Interface Inheritance vs. Implementation Inheritance**
  - **Interface Inheritance**
    - Inherit only **method signatures**
    - E.g., **interfaces** in Java
  - **Implementation Inheritance**
    - Inherit everything: **method signatures**, **method implementations**, and **fields**
    - E.g., **classes** (in any object-oriented programming languages)

- **Single Inheritance in Java**
  - A class (**Child**) inherits (implementation) from another class (**Parent**)
    - **class Child extends Parent**

- **Multiple Inheritance in Java**
  - A class (**Impl**) inherits (interface) from interfaces (**Inter1**, **Inter2**)
    - **class Impl implements Inter1, Inter2**
  - An interface (**Inter0**) inherits from other interfaces (**Inter1**, **Inter2**)
    - **interface Inter0 extends Inter1, Inter2**

- **Single Inheritance (on class): Parent and Child**
  - The **parent** class is the **direct/immediate superclass** of its **children**.
  - The **child** class is a **direct/immediate subclass** of its **parent**.
  - Each **ancestor** (parent of parent of …) of a class is its **superclass**.
  - Each **descendant** (children of children of …) of a class is its **subclass**.
- **Class Modifiers (other than accessible modifiers)**
  - **Final** – cannot have a subclass
  - **Abstract** – cannot be instantiated
    (i.e., no object; should have a subclass)
- **Field/Method Modifiers**
  - **Final** – cannot be modified; cannot be overridden
  - **Abstract** (**method**) – need to be implemented in subclasses

- **(Interface/Implementation) Inheritance and Type Substitutability**
  - Supertype and Subtype
    - If (class/interface) **Child** **inherits** **from** (class/interface) **Parent**, then
      - (class/interface) **Child** **is a** **subtype** **of** (class/interface) **Parent** and
      - (class/interface) **Parent** **is a** **supertype** **of** (class/interface) **Child**
      - Any object of type **Child** **can** **behave like** an object of type **Parent**
      - Type **Parent** can always to **be** **substituted** **by** Type **Child**
- **SOLID principles**
  - **Single Responsibility**: Every type should have only one responsibility
  - **Open–Closed**: Open for extension but Closed for modification
  - **Liskov Substitution**: A supertype must be replaceable by its subtypes without knowing
  - **Interface Segregation**: Many specific interfaces are better than one general interface
  - **Dependency Inversion**: Depend upon abstractions, not concretions

• Inheritance and Constructor in Java

- Constructors are not inherited
  - **Constructors are not members of a class**
    - unlike fields, methods, nested classes which are members and inherited
- The constructor without parameters is called **Empty Constructor**
  - Empty constructors are **the default constructors** of every class that the compiler inserts into each class automatically and implicitly if there are no constructors at all in that class.
  - All constructors of any class will **implicitly and automatically** make ***the first call*** to the empty constructors of their superclass, i.e., **super();**, OR **explicitly** make ***the first call*** to one constructor of their superclass, i.e., **super(…);**
  - All calls to any constructors of its superclass must be ***the first call*** *implicitly or explicitly* in the constructor of the subclass.

- **Interface Accessibility**
  - Public Interface – accessible by all
  - Package-Private Interface – accessible within the package only
- **Members of Interfaces:**
  - **methods, fields, (nested) classes, nested interfaces**
  - All **fields/attributes** are ***public static*** and ***final***, i.e., **constants**
  - All **(*abstract*/default/static) methods are *public*** (implicitly or explicitly)
  - Abstract methods have no body but default and static methods
- **Types of Interfaces**
  - **Normal Interfaces**
    - Contain one or more methods.
  - **Marker Interfaces**
    - Contain nothing. E.g., Serializable interface
  - **Functional Interfaces … @FunctionalInterface**
    - Contain only one abstract method; all other methods must not be abstract