

# OOP in Java

---



**Fawzi Emad**  
**Chau-Wen Tseng**

**Department of Computer Science**  
**University of Maryland, College Park**

## Object Oriented Programming (OOP)

- **OO Principles**
  - **Abstraction**
  - **Encapsulation**
- **Abstract Data Type (ADT)**
  - **Implementation independent interfaces**
  - **Data and operations on data**
- **Java**
  - **Many language features supporting OOP**

## Overview

- **Objects & class**
- **References & alias**
- **“this” & “super” reference**
- **Constructor**
- **Garbage collection & destructor**
- **Modifiers**
  - **Public, Private, Protected**
  - **Static**
  - **Final**

## Object & Class

- **Object**
  - **Abstracts away (data, algorithm) details**
  - **Encapsulates data**
  - **Instances exist at run time**
- **Class**
  - **Blueprint for objects (of same type)**
  - **Exists at compile time**

## References & Aliases

### ■ Reference

- A way to get to an object, not the object itself
- All variables in Java are **references** to objects

### ■ Alias

- Multiple references to same object
- “X == Y” operator tests for alias
- X.equals(Y) tests contents of object (potentially)



## References & Aliases – Issues

### ■ Copying

#### ■ References

```
X = new Object();  
Y = X;           // Y refers to same object as X
```

#### ■ Objects

```
X = new Object();  
Y = X.clone();   // Y refers to different object
```

### ■ Modifying objects

```
X = new Object();  
Y = X;  
X.change();      // modifies object for Y
```

## “this” Reference

### ■ Description

- Reserved keyword
- Refers to object through which method was invoked
- Allows object to refer to itself
- Use to refer to instance variables of object

## “this” Reference – Example

```
class Node {  
    value val1;  
    value val2;  
    void foo(value val2) {  
        ... = val1;           // same as this.val1 (implicit this)  
        ... = val2;           // parameter to method  
        ... = this.val2;      // instance variable for object  
        bar( this );          // passes reference to object  
    }  
}
```

# Inheritance

- **Definition**
  - Relationship between classes when state and behavior of one class is a subset of another class
- **Terminology**
  - Superclass / parent ⇒ More general class
  - Subclass ⇒ More specialized class
- **Forms a class hierarchy**
- **Helps promote code reuse**

## “super” Reference

- **Description**
  - Reserved keyword
  - Refers to superclass
  - Allows object to refer to methods / variables in superclass
- **Examples**
  - `super.x` // accesses variable x in superclass
  - `super()` // invokes constructor in superclass
  - `super.foo()` // invokes method foo() in superclass

# Constructor

## ■ Description

- Method invoked when object is instantiated
- Helps initialize object
- Method with same name as class **w/o** return type
- Implicitly invokes constructor for superclass
  - If not explicitly included

## Constructor – Example

```
class foo {  
    foo() { ... }           // constructor for foo  
}  
class bar extends foo {  
    bar() {                 // constructor for bar  
                           // implicitly invokes foo() here  
        ...  
    }  
}  
class bar2 extends foo {  
    bar2() {                // constructor for bar  
        super();           // explicitly invokes foo() here  
    }  
}
```

# Garbage Collection

## ■ Concepts

- All interactions with objects occurs through reference variables
- If no reference to object exists, object becomes **garbage** (useless, no longer affects program)

## ■ Garbage collection

- Reclaiming memory used by unreferenced objects
- Periodically performed by Java
- Not guaranteed to occur
- Only needed if running low on memory

# Destructor

## ■ Description

- Method with name **finalize()**
- Returns void
- Contains action performed when object is freed
- Invoked automatically by garbage collector
  - Not invoked if garbage collection does not occur
- Usually needed only for non-Java methods

## ■ Example

```
class foo {  
    void finalize() { ... }    // destructor for foo  
}
```

## Method Overloading

### ■ Description

- Same name refers to multiple methods

### ■ Sources of overloading

- Multiple methods with different parameters
  - Constructors frequently overloaded
- Redefine method in subclass

### ■ Example

```
class foo {  
    foo() { ... }           // constructor for foo  
    foo(int n) { ... }     // 2nd constructor for foo  
}
```

## Modifier

### ■ Description

- Java keyword (added to definition)
- Specifies characteristics of a language construct

### ■ (Partial) list of modifiers

- Public / private / protected
- Static
- Final
- Abstract

## Modifier

### ■ Example

```
public class foo {  
    private static int count;  
    private final int increment = 5;  
    protected void finalize { ... }  
}  
public abstract class bar {  
    abstract int go() { ... }  
}
```

## Visibility Modifier

### ■ Properties

- Controls access to class members
- Applied to instance variables & methods

### ■ Types

- Public
  - May be directly referenced **outside** object
- Private
  - Referenced only **within** class definition
- Protected
  - Referenced within class definition & **by subclasses**

## Visibility Modifier

- **For instance variables**
  - Should usually be **private** to enforce encapsulation
  - Sometimes may be **protected** for subclass access
- **For methods**
  - **Public methods** – provide services to clients
  - **Private methods** – provide support other methods
  - **Protected methods** – provide support for subclass

## Modifier – Static

- **Static variable**
  - Single copy for class
  - Shared among all objects of class
- **Static method**
  - Can be invoked through class name
  - Does not need to be invoked through object
  - Can be used even if no objects of class exist
  - Can not reference instance variables

## Modifier – Final

- **Final variable**
  - Value can not be changed
  - Must be initialized in every constructor
  - Attempts to modify final are caught at compile time
- **Final static variable**
  - Used for constants
  - Example
    - `final static int Increment = 5;`

## Modifier – Final

- **Final method**
  - Method **can not be overloaded** by subclass
  - Private methods are implicitly final
- **Final class**
  - Class can not be a superclass (extended)
  - Methods in final class are implicitly final

## Modifier – Final

- **Using final classes**
  - Prevents inheritance / polymorphism
  - May be useful for
    - Security
    - Object oriented design
- **Example – class `String` is final**
  - Programs can depend on properties specified in Java library API
  - Prevents subclass that may bypass security restrictions

## Modifier – Abstract

- **Description**
  - Represents generic concept
  - Can not be instantiated
- **Abstract class**
  - Placeholder in class hierarchy
  - Can be partial description of class
  - Can contain non-abstract methods
  - Required if any method in class is abstract
- **Example**

```
abstract class foo {           // abstract class
    abstract void bar() { ... } // abstract method
```

# Interface

- **Description**

- **Collection of**

- **Constants**

- **Abstract methods**

- **Can not be instantiated**

- **Classes can **implement** interface**

- **Must implement **all** methods in interface**

- **Example**

- ```
class foo implements bar { ... } // interface bar
```