Software Practice 1 - Inheritance and Interface

- Inheritance
- Overriding
- Polymorphism
- Abstraction
- Encapsulation
- Interfaces

Prof. Joonwon Lee

T.A. Jaehyun Song
Jongseok Kim (42)

T.A. Sujin Oh
Junseong Lee (43)

In Previous Lecture...

- **We learned about OOP which projects the real world in programming language**
- **Also we learned about**
  - Class: A structure that defines fields and methods
  - Object: An executable copy of a class
- **And each object can**
  - encapsulate some fields or methods
  - be referred by or refer to other objects

Inheritance

- Inheritance can be defined as the process where **one class acquires the properties (methods and fields) of another.** With the use of inheritance the information is made manageable in a hierarchical order.
- The class which **inherits** the properties of other is known as **subclass (derived class, child class)** and the class whose properties **are inherited** is known as **superclass (base class, parent class).**

extends Keyword

- **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.
- **Syntax**

```
class Super {
        .....
        .....
}
class Sub extends Super {
        .....
        .....
}
```

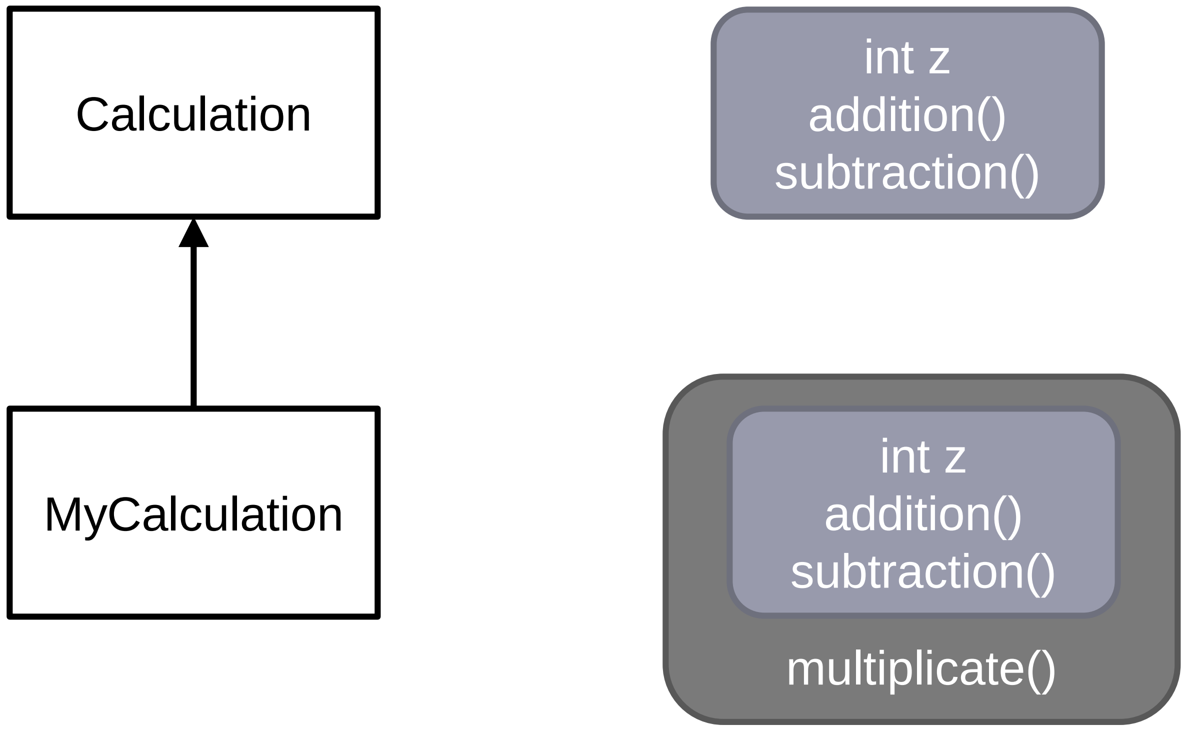
Inheritance Example

```
class Calculation {
  int z;
  public void add(int x, int y) {
         z = x + y;
        System.out.println("The sum of the given numbers:"+z);
  }
  public void subtract(int x, int y) {
        z = x - y;
        System.out.println("The difference between the given numbers:"+z);
  }
}

public class MyCalculation extends Calculation {

  public void multiplicate(int x, int y) {
        z = x * y;
        System.out.println("The product of the given numbers:"+z);
  }
  public static void main(String[] args) {
        int a = 20, b = 10;
        MyCalculation demo = new MyCalculation();
        demo.add(a, b);
        demo.subtract(a, b);
        demo.multiplicate(a, b);
  }
}
```

Example – cont'd



super - Keyword

- The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.
  - It is used to differentiate the members of superclass from the members of subclass, if they have same names.
  - It is used **to invoke the superclass constructor from subclass**.
- If a class is inheriting the properties of another class, and if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable
super.method();
```

super - Example

```java
class SuperClass {
  int num = 20;
  // display method of superclass
  public void display() {
    System.out.println("This is the display method of superclass");
  }
}
public class SubClass extends SuperClass {
  int num = 10;
  // display method of subclass
  public void display() {
    System.out.println("This is the display method of subclass");
  }
  public void myMethod() {
    this.display(); // Invoking the display() method of sub class
    super.display(); // Invoking the display() method of superclass

    // printing the value of variable num of subclass
    System.out.println("value of the variable named num in sub class:"+ sub.num);
    // printing the value of variable num of superclass
    System.out.println("value of the variable named num in super class:"+ super.num);
  }
  public static void main(String[] args) {
    SubClass obj = new SubClass();
    obj.myMethod();
  }
}
```

super - Invoke Constructor

- If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below.

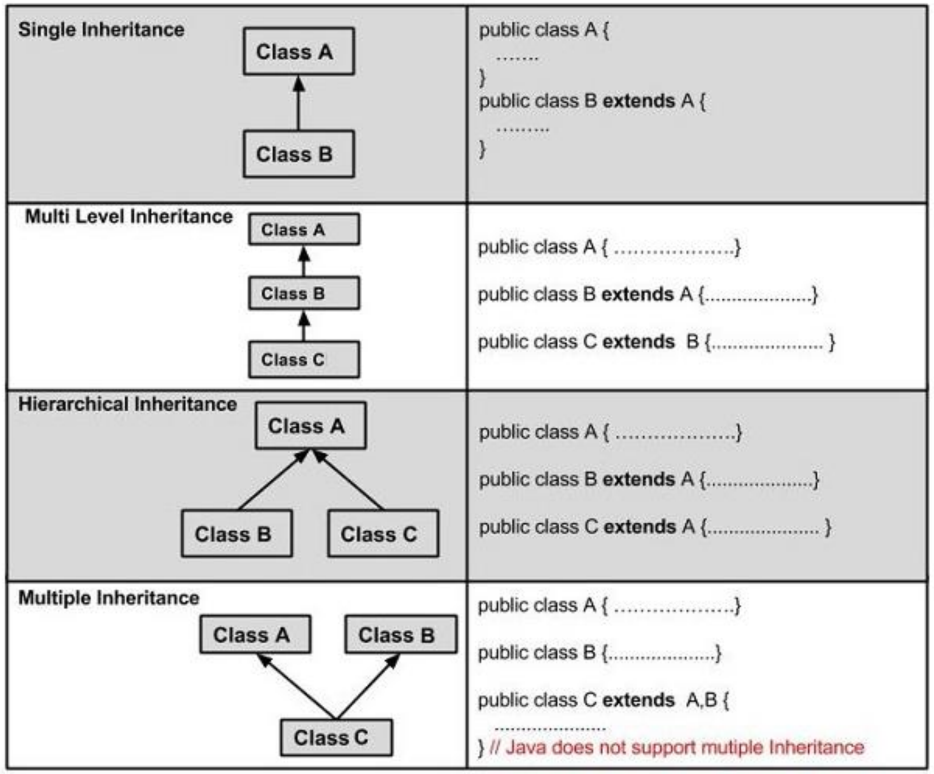
```
super(values);
```

super - Invoke Constructor

```
class SuperClass {
  int age;
  SuperClass(int age) {
      this.age = age;
  }
  public void getAge() {
      System.out.println("The value of the variable" +
                         " named age in super class is: " + age);
  }
}
public class SubClass extends SuperClass {
  SubClass(int age) {
      super(age);
  }
  public static void main(String[] args) {
      SubClass s = new SubClass(24);
      s.getAge();
  }
}
```

Type of Inheritance

- There are various types of inheritance as demonstrated below.



Single Inheritance	public class A {
	……..
	}
	public class B **extends** A {
	………
	}

Multi Level Inheritance	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** B {………………. }

Hierarchical Inheritance	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** A {………………. }

Multiple Inheritance	public class A { ……………….}
	public class B {……………….}
	public class C **extends** A,B {
	……………….
	} // Java does not support mutiple Inheritance

Class A
Class B
Class A
Class B
Class C
Class A
Class B
Class C
Class A
Class B
Class C

Overriding

- **Method overriding**, in object oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super classes or parent classes. The implementation in the subclass overrides (replaces) the implementation in the superclass by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class.

Overriding - Example

```
class Animal {
   public void move() {
            System.out.println("Animals can move");
   }
}
class Dog extends Animal {
   public void move() {
            System.out.println("Dogs can walk and run");
   }
}
public class TestDog {
   public static void main(String args[]) {
            Animal a = new Animal(); // Animal reference and object
            Animal b = new Dog(); // Animal reference but Dog object
            a.move(); // runs the method in Animal class
            b.move(); // runs the method in Dog class
   }
}
```

Rules for Method Overriding

- The argument list and return type should be same with the overridden method.
- The access level cannot be more restrictive than the overridden method's.
  - If superclass method is declared public, subclass method cannot be either private or protected.
- A method declared final cannot be overridden.
- A method declared static is always re-declared.
- Constructors cannot be overridden.

final - Keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
  - Variable – stop value change
  - Method – stop method overriding
  - Class – stop inheritance

final - Example

```
final class FinalClass {
  final int finalVariable;
  final public void move() {
    finalVariable = 2;      // ERROR
    System.out.println(“I’m final method");
  }
}
class SubClass extends FinalClass {  //ERROR
  public void move() {               //ERROR
    System.out.println(“Method overriding");
  }
}
public class TestDog {
  public static void main(String args[]) {
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object
    a.move(); // runs the method in Animal class
    b.move(); // runs the method in Dog class
  }
}
```

static - Keyword

- **Applies to fields and methods**
- **Means that the field/method**
  - is defined for the class declaration,
  - is not unique for each instance

static - Example

- **Keep track of the number of babies that have been made.**

```
public class Baby {
  int numBabiesMade = 0;
  Baby() {
    numBabiesMade += 1;
  }
}


Baby b1 = new Baby();
Baby b2 = new Baby();
System.out.println (b1.numBabiesMade);      // 1
System.out.println (b2.numBabiesMade);      // 1
System.out.println (Baby.numBabiesMade);    // Error!
```

static - Example

- **Keep track of the number of babies that have been made.**

```
public class Baby {
  static int numBabiesMade = 0;
  Baby() {
    numBabiesMade += 1;
  }
}

Baby b1 = new Baby();
Baby b2 = new Baby();
System.out.println (b1.numBabiesMade);      // 2
System.out.println (b2.numBabiesMade);      // 2
System.out.println (Baby.numBabiesMade);    // 2
```

Polymorphism

- Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a **super class reference is used to refer to a sub class object.**

Polymorphism - Instantiate

```
class Animal { ... }
class Lion extends Animal { ... }
class Dog extends Animal { ... }
class Cat extends Animal { ... }
class Wolf extends Animal { ... }
```

```
Animal[] animals = new Animal[4];
animals[0] = new Dog();
animals[1] = new Lion();
animals[2] = new Cat();
animals[3] = new Cat();
```

Polymorphism - Overloading

- Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different. In the last class we discussed constructor overloading that allows a class to have more than one constructors having different argument lists.
- **Argument lists could differ in –**
  1. Number of parameters.
  2. Data type of parameters.
  3. Sequence of Data type of parameters.
- **Method overloading** is also known as **Static Polymorphism**.

Method Overloading

```
int add (int x, int y) {
        return x+y;
}
float add (float x, float y) {
        return x+y;
}
// available codes!
```

Method Overloading

```
int add (int x, int y) {
        return x+y;
}
float add (float x, float y) {
        return x+y;
}
int add (int A, int B) { … } ??
int add (int A, int B, float C) { … } ??
int add (float x, int y1, int y2) { … } ??
```

Method Overloading

```
int add (int x, int y) {
        return x+y;
}
float add (float x, float y) {
        return x+y;
}
int add (int A, int B) { … } ERROR
int add (int A, int B, float C) { … } O
int add (float x, int y1, int y2) { … } O
```

Special Classes

ABSTRACT AND INTERFACE

Abstraction in PL

- **Abstraction** occurs during class level design, with the objective of **hiding the implementation complexity** of **how** the features offered by an API / design / system were implemented, in a sense simplifying the 'interface' to access the underlying implementation.

*PL: Programming Language

Encapsulation

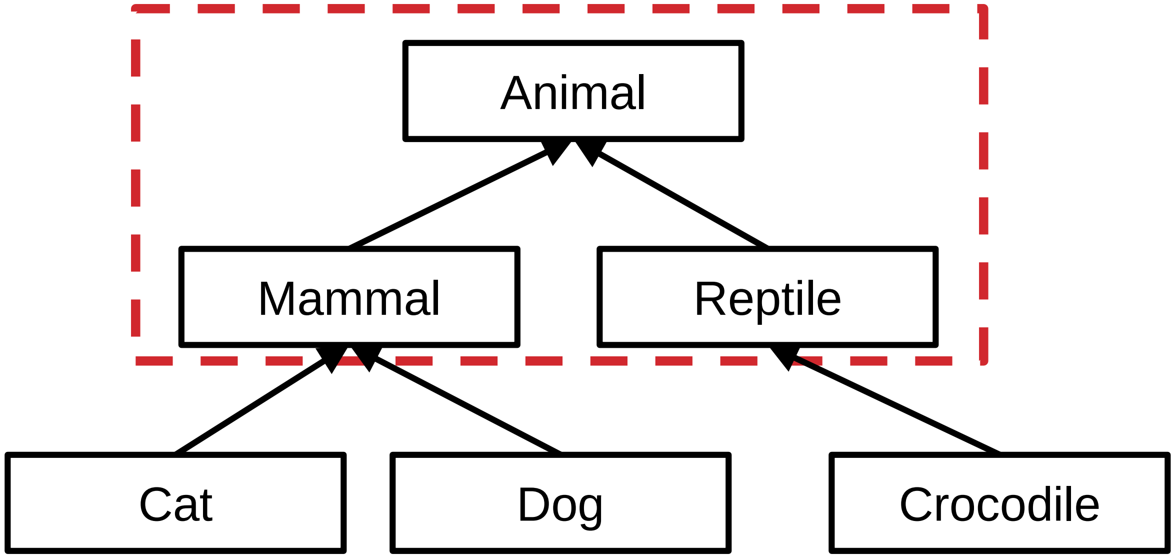
- **Encapsulation**, often referred to as "**Information Hiding**", revolves more specifically around the hiding of the internal data (e.g. state) owned by a class instance, and enforcing access to the internal data in a controlled manner.

Abstract Class

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
  - Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
  - But, if a class has at least one abstract method, then the class must be declared abstract.
  - **If a class is declared abstract, it cannot be instantiated.**
  - To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
  - If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract Class

- **Assume that the class hierarchy as follow :**



I don’t want to instantiate Mammal and Reptile!
Then, declare those classes using abstract keyword.

Mammal[ ] mammalList = new Mammal[5]; ?

Abstract Method

- If you want a class to contain a particular method but you want the actual implementation of that **method to be determined by child classes**, you can declare the method in the parent class as an abstract.
  - abstract keyword is used to declare the method as abstract.
  - You have to place the abstract keyword before the method name in the method declaration.
  - An abstract method contains a method signature, but no method body.
  - Instead of curly braces, an abstract method will have a semi colon (;) at the end.

Abstract Method - Example

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;
   public abstract double computePay(); // Remainder of class definition
}
```

- Declaring a method as abstract has two consequences
  - The class containing it must be declared as abstract.
  - Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Interface

- An interface is a reference type in Java. It is similar to class. **It is a collection of abstract methods (and static/final variables)**. A class implements an interface, thereby inheriting the abstract methods of the interface.

Interface vs Class

- **An interface is similar to a class**
  - An interface can contain any number of methods.
  - An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
  - The byte code of an interface appears in a .class file.
- **An interface is different from a class**
  - You cannot instantiate an interface.
  - An interface does not contain any constructors.
  - All of the methods in an interface are abstract; **(do not need use abstract keyword)**
  - An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - **An interface is not extended by a class; it is implemented by a class.**
  - An class can implement multiple interfaces.
  - An interface can extend multiple interfaces.

Implements keyword

- A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
public class MammalInt implements Animal {
        public void eat() {
                System.out.println("Mammal eats");
        }
        public void travel() {
                System.out.println("Mammal travels");
        }
                public int noOfLegs() {
                return 0;
        }
        public static void main(String args[]) {
                MammalInt m = new MammalInt();
                m.eat();
                m.travel();
        }
}
```

Implementing Multiple Interfaces

- A Java class can implement multiple interfaces.
- The implements keyword is used once, and interfaces are declared in a comma-separated list.

```
public class Cat implements Animal, Master
```

Extends keyword

- An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

```
// Filename: Sports.java
public interface Sports {
	public void setHomeTeam(String name);
	public void setVisitingTeam(String name);
}
// Filename: Football.java
public interface Football extends Sports {
	public void homeTeamScored(int points);
	public void visitingTeamScored(int points);
	public void endOfQuarter(int quarter);
}
// Filename: Hockey.java
public interface Hockey extends Sports {
	public void homeGoalScored();
	public void visitingGoalScored();
	public void endOfPeriod(int period);
	public void overtimePeriod(int ot);
}
```

Extending Multiple Interfaces

- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
- The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

```
public interface Hockey extends Sports, Event
```

[Lab – Practice #3]

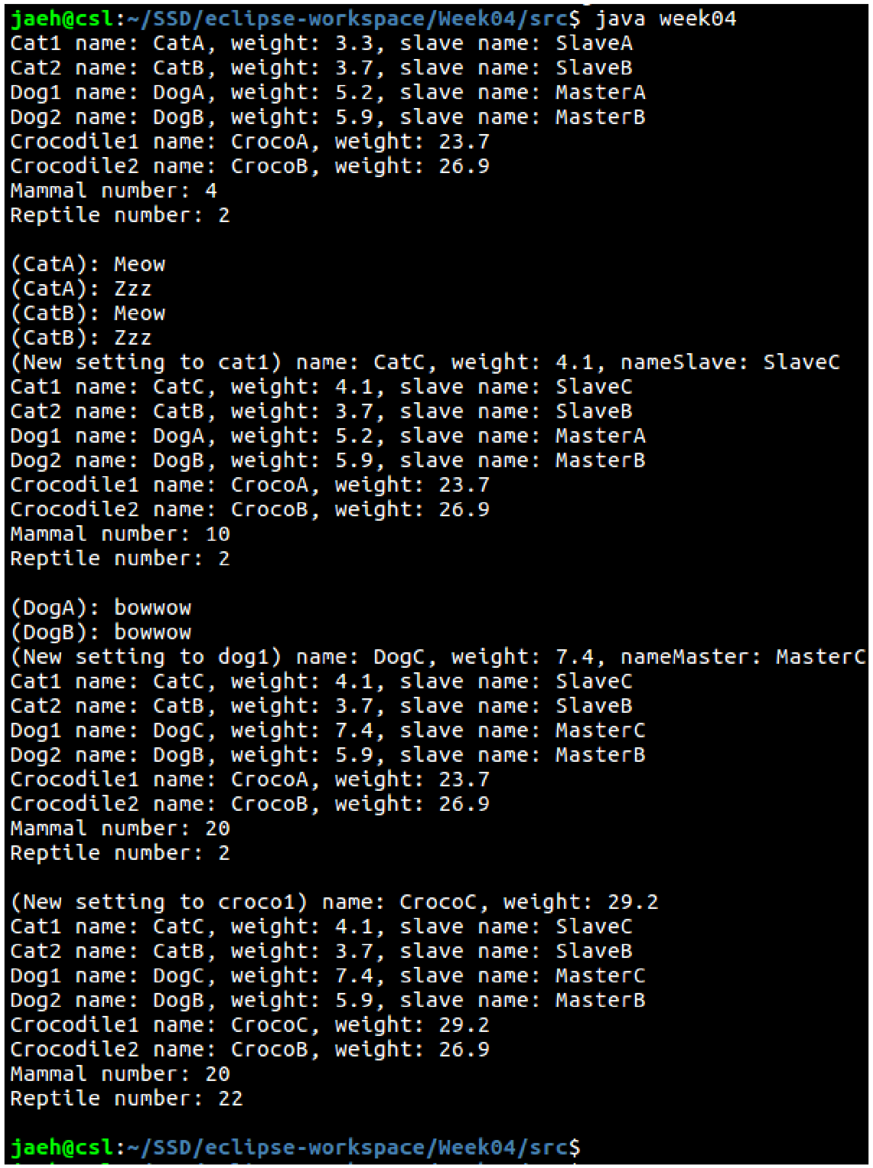
- **To satisfy the requirements, design class hierarchy and implement "Animal.java"**

- **Class Animal // super class**
  - Fields : String name, float weight;
  - Methods : get/set each fields
- **Abstract class Mammal extends Animal**
  - Fields : int numMammal;
  - Methods : int getNumMammals();
- **Abstract class Reptile extends Animal**
  - Fields : int numReptile;
  - Methods : int getNumReptile;
- **Class Cat extends Mammal**
  - Fields : String nameSlave;
  - Methods : get/set each fields, meow(), sleep(), breed();
- **Class Dog extends Mammal**
  - Fields : String nameMaster;
  - Methods : get/set each fields, bark(), breed();
- **Class Crocodile extends Reptile**
  - Methods : get/set each fields, spawn();

[Lab – Practice #3]

- **Class Cat, Dog, Crocodile**
  - Should be final class
  - Constructor mandatory initiate name and optionally initiate weight
- **Class Cat**
  - breed() : breed three kitties // should be implemented
  - meow() : print message “(name) : meow”
  - sleep() : print message “(name) : Zzz”
- **Class Dog**
  - breed() : breed five puppies // should be implemented
  - bark() : print message “(name) : bowwow”
- **Class Crocodile**
  - spawn() : spawn twenty eggs

[Lab – Practice #3]



```
jaeh@csl:~/SSD/eclipse-workspace/Week04/src$ java week04
Cat1 name: CatA, weight: 3.3, slave name: SlaveA
Cat2 name: CatB, weight: 3.7, slave name: SlaveB
Dog1 name: DogA, weight: 5.2, slave name: MasterA
Dog2 name: DogB, weight: 5.9, slave name: MasterB
Crocodile1 name: CrocoA, weight: 23.7
Crocodile2 name: CrocoB, weight: 26.9
Mammal number: 4
Reptile number: 2

(CatA): Meow
(CatA): Zzz
(CatB): Meow
(CatB): Zzz
(New setting to cat1) name: CatC, weight: 4.1, nameSlave: SlaveC
Cat1 name: CatC, weight: 4.1, slave name: SlaveC
Cat2 name: CatB, weight: 3.7, slave name: SlaveB
Dog1 name: DogA, weight: 5.2, slave name: MasterA
Dog2 name: DogB, weight: 5.9, slave name: MasterB
Crocodile1 name: CrocoA, weight: 23.7
Crocodile2 name: CrocoB, weight: 26.9
Mammal number: 10
Reptile number: 2

(DogA): bowwow
(DogB): bowwow
(New setting to dog1) name: DogC, weight: 7.4, nameMaster: MasterC
Cat1 name: CatC, weight: 4.1, slave name: SlaveC
Cat2 name: CatB, weight: 3.7, slave name: SlaveB
Dog1 name: DogC, weight: 7.4, slave name: MasterC
Dog2 name: DogB, weight: 5.9, slave name: MasterB
Crocodile1 name: CrocoA, weight: 23.7
Crocodile2 name: CrocoB, weight: 26.9
Mammal number: 20
Reptile number: 2

(New setting to croco1) name: CrocoC, weight: 29.2
Cat1 name: CatC, weight: 4.1, slave name: SlaveC
Cat2 name: CatB, weight: 3.7, slave name: SlaveB
Dog1 name: DogC, weight: 7.4, slave name: MasterC
Dog2 name: DogB, weight: 5.9, slave name: MasterB
Crocodile1 name: CrocoC, weight: 29.2
Crocodile2 name: CrocoB, weight: 26.9
Mammal number: 20
Reptile number: 22

jaeh@csl:~/SSD/eclipse-workspace/Week04/src$
```

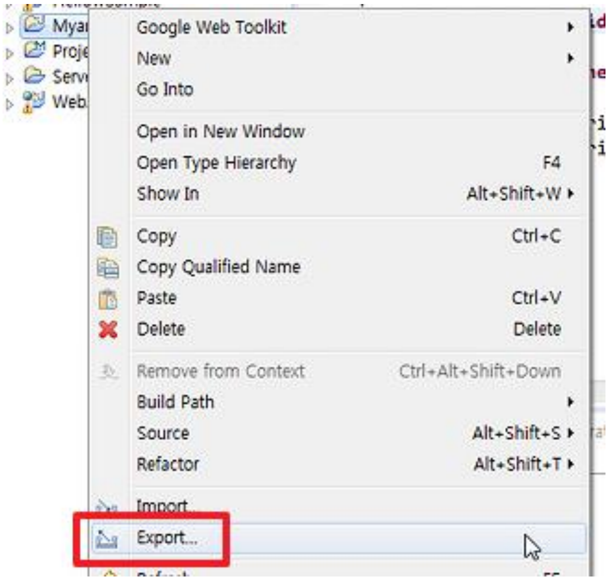
- **Skeleton code is uploaded on i-Campus**
- **Fill in the all class except to week04**
- **Left image is the output**
  - Your answer should be same

[Submit]

- **Upload to i-Campus**
  - Compress your project directory to zip file
  - File name: studentID_lab03.zip

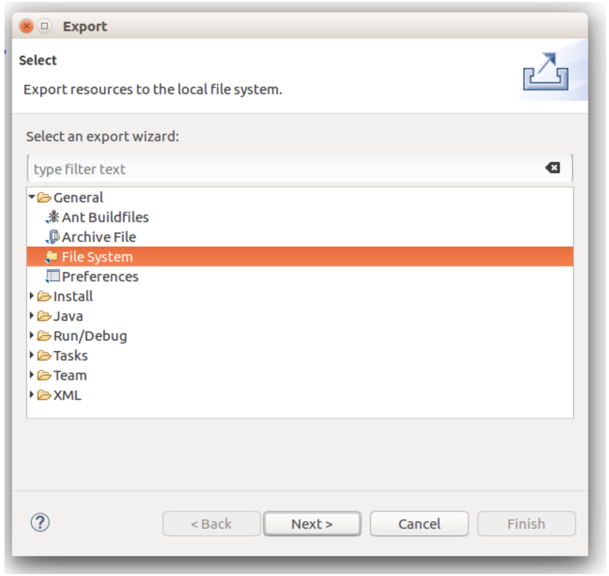
- **Due date**
  - Today 23:59:59
    - Class 42 (3/26 Monday)
    - Class 43 (3/28 Wednesday)
  - Penalty: -10% of each lab score per one day

[Project Export]



1. Click mouse right button at project you want to export & choose Export

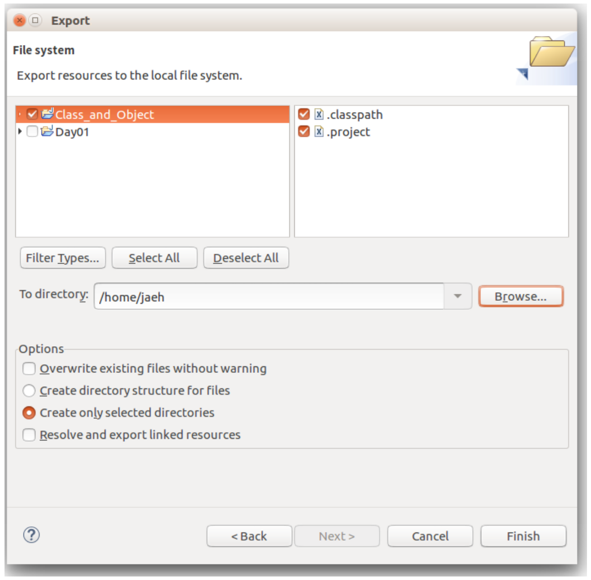
2. Choose ***General > File system*** and click next button


Export

Select

Export resources to the local file system.

Select an export wizard:


Export

File system

Export resources to the local file system.

[✓] Class_and_Object
[ ] Day01
[✓] .classpath
[✓] .project

To directory: /home/jaeh

Options
- [ ] Overwrite existing files without warning
- ( ) Create directory structure for files
- (●) Create only selected directories
- [ ] Resolve and export linked resources

3. Designate save directory path and click Finish button