PROGRAMMING UNIT 4 - OBJECTS & CLASSES & INHERITANCE

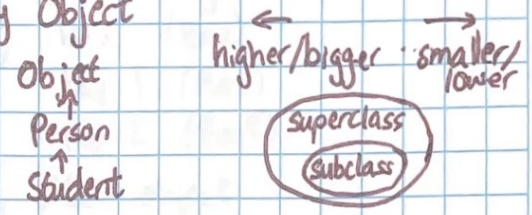
01 Class Hierarchies

- inherits - subclass inherits from superclass, subclass extends superclass
- All classes inherit from Object
- e.g. class Samoyed extends Dog { ____ }
  e.g. class Fraction
  class Fraction extends Object] equivalent
- When a class inherits from another, it gets all of its fields & methods.
- When a method is redifined in a subclass, it is overriden.

02 Terminology - printed sheets

03 Inheritance & Methods & Variables (both files)

- equals and toString are provided by Object
- Valid:
  - Student s = new Student();
  - Person p = new Student();
  - Object o = new Student();
- Invalid:
  - Object o = new Object(); Student s = o;
  - Object o = new Student(); Student s = o; ✗ Student s = (Student)o; ✓



- Inheriting Fields:
  - Private fields are still private to a class.
  - To use/see/manipulate inherited private values, you must use accessor and mutator methods
  - Protected makes the field visible to any class which inherits (subclasses), but not above in the hierarchy, nor outside the hierarchy.

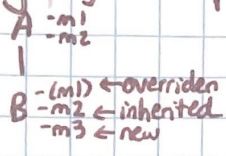
04 Using Super

- It is possible to refer explicitly to the superclass of an object using super
- class Student extends Person{
  // super(); ← happens in background
  private String number;
  public Student(String name, String number){
  super(name); // like Person(name)
  this.number = number;
  }
  }
- Constructors:
  - call super() at top
- Other Methods:
  - super.methodName();
  e.g. super.toString();

05 Employee Assignment

06 Polymorphism

- Arrays of a superclass can store any subtype as elements
-



A var = new B();
Σ has: -(m1) ← overriden
-m2 ← inherited

B var2 = new B();
Σ has: (m1), m2, m3

- examples:

EXAMPLE 1 (Regular)

CLASS

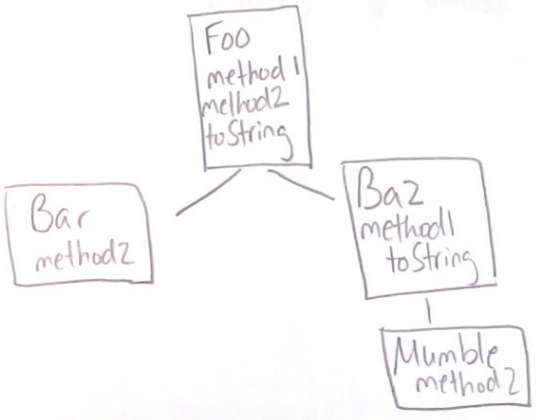
```
public class Foo{
    public void method1(){
        System.out.println("foo1");
    }
    public void method2(){
        System.out.println("foo2");
    }
    public String toString(){
        return "foo";
    }
}
public class Bar extends Foo{
    public void method2(){
        System.out.println("bar 2");
    }
}
public class Baz extends Foo{
    public void method1(){
        System.out.println("baz 1");
    }
    public String toString(){
        return "baz";
    }
}
public class Mumble extends Baz{
    public void method2(){
        System.out.println("mumble 2");
    }
}
```

CLIENT

```
Foo[] pity = {new Baz(), new Bar(),
              new Mumble(), new Foo()};

for(int i=0; i<pity.length; i++){
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

OUTPUT

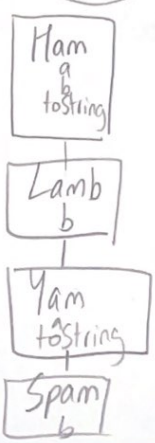
```
// baz object
baz (Baz)
baz 1 (Baz)
foo 2 (Foo↑)
// bar object
foo (Foo↑)
foo 1 (Foo↑)
bar 2 (Bar)
// mumble object
baz (Baz↑)
baz 1 (Baz↑)
mumble 2 (Mumble)

// foo object
foo (Foo)
foo 1 (Foo)
foo 2 (Foo)
```



Example 2 (Super/other calls)

CLASS

```
public class Lamb extends Ham{
    public void b(){
        System.out.print("Lamb b");
    }
}
public class Ham{
    public void a(){
        System.out.print("Ham a");
        b();
    }
    public void b(){
        System.out.print("Ham b");
    }
    public String toString(){
        return "Ham";
    }
}
public class Spam extends Yam{
    public void b(){
        System.out.print("Spam b");
    }
}
public class Yam extends Lamb{
    public void a(){
        System.out.print("Yam a");
        super.a();
    }
    public String toString(){
        return "Yam";
    }
}
```



CLIENT

```
Ham[] food={new Lamb(), new Ham(), new
        Spam(), new Yam()};
for(int i=0; i< food.length; i++){
    System.out.println(food[i]);
    food[i].a();
    System.out.println();
    food[i].b();
    System.out.println();
}
```

OUTPUT

```
// Lamb Object
Ham (Ham↑)
Ham a Lamb b (Ham↑, Lamb)
Lamb b (Lamb)

// Ham Object
Ham (Ham)
Ham a Ham b (Ham)
Ham b (Ham)

// Spam Object
Yam (Yam↑)
Yam a Ham a Spam b (Yam↑, Ham↑, S
Spam b (Spam)

// Yam Object
Yam (Yam)
Yam a Ham a Lamb b (Yam, Ham↑, Lamb
Lamb b (Lamb↑)
```

Introduction to Inheritance and Polymorphism

- **Inheritance** extends on an existing class
- **Polymorphism** is something you can do to classes that are being inherited
- You might want to extend a class by adding more specific items into the object
    - A car class might be extended to "BMW" class where it has BMW specific methods and variables
    - Or a Circle class might be extended to "Disk" and accepts the input of height.

Inheritance

- **Inheritance:** when you extend classes from a "super class" to make it do more specific things
    - "allows a class to define a specialized type of an already existing class"
    - **Is-a relationship:** an extended class "is a" type of another class
        - BMW "is a" type of a car class. Disk "is a" type of circle class.
- **Implementation:**
    - **extends:** a keyword you must use to declare a "sub class"
        - this keyword will make it refer to the super class
        - public class <name> extends <superclass_name> { … }
    - **base class:** or "super class: is the class which this sub class is based on
    - **derived class:** or "sub class" often contains overridden methods or new data from the base class
    - **super:** is a keyword used to access methods in the base class
        - Ex. super(r) from the BMW class will call the constructor of the super class (car) , and pass variable r into it.
    - Members that are **private** cannot be accessed by derived classes
        - Hence, accessor methods are used to get values (ie getWeight(), getRadius())
    - Inherited Methods can be called directly.
        - If M3 is the object of BMW class, and you call carStart() from client code. If BMW class doesn't have it's own carStart() method, but Car class does, since BMW inherited the Car class, the client code doesn't care and still executes carStart().

Polymorphism

- **Polymorphism:** an OOP property where objects have the ability to assume different types.
    - Polymorphism is based on inheritance
    - Since a subclass is derived from a super class, a super class object can reference an object of the subclass

ie Car honda;

BMW 320i = new BMW(<arguments>);
honda = 320i //honda now references 320i

- Since BMW is inherited from Car class, any object made from Car can be "morphed" into another object made by a subclass of Car, in this case, a BMW 320i object.

- Overridden methods from the super class can be called from the sub class.
- From the example, the method toString() originally in Car was overridden in BMW. Even though honda was made by Car class, then later morphed into a BMW object, I can still call the toString() method overridden by BMW.