03. Class concepts – (Java)

Class fundamentals
Methods
Constructors
Inner Classes

Method Overloading

- Defining two or more methods within the same class that share the same name, as long as their parameter declarations are different is called method overloading.
- This is the way Java implements polymorphism.

```
class OverloadDemo  {

    void test() {
            System.out.println("No parameters");
    }

    void test(int a) {
            System.out.println("a: " +a);
    }
                                        // double test(int a) {. . . } - Wrong
    void test(int a, int b) {
            System.out.println("a and b: " +a + " " +b);
    }

    double test(double a) {
            System.out.println("double a: " +a);
            return a*a;
    }
}
```

```
class Overload  {
   public static void main(String args[])  {
        OverloadDemo ob = new OverloadDemo();
        double result;

        ob.test();
        ob.test(10);
        ob.test(10, 20);
        result = ob.test(123.25);
        System.out.println("Result of ob.test(123.25): "
                                  +result);
   }
}
```

Constructor Overloading

- Constructors can also be overloaded.

```
Box(double w, double h, double d)  {
        width = w;   height = h;   depth = d;
}

Box()  {
        width = -1;   height = -1;   depth = -1;
 }

Box(double len)  {
        width = height = depth = len;
}
```

'this' keyword

Local variables

```
Box(double width, double height, double depth)  {
        this.width = width;
        this.height = height;
        this.depth = depth;
}
```

Instance variables

Static Members

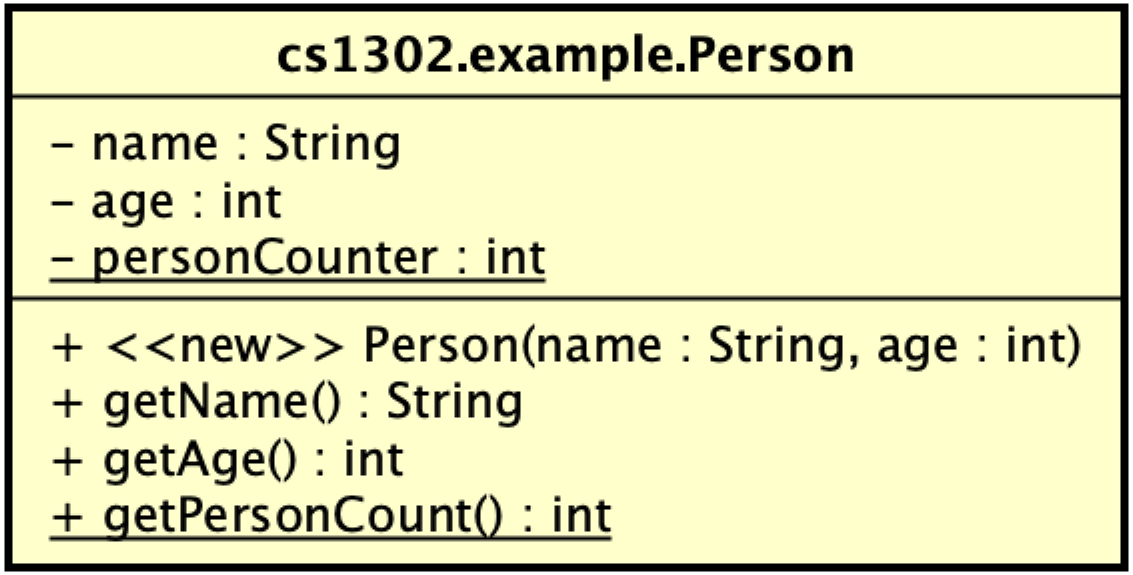
- A static class member can be accessed directly by the class name and doesn’t need any object. A single copy of a static member is maintained throughout the program regardless of the number of objects created.
- Static variables are initialized only once and at the start of the execution during the lifetime of a class. These variables will be initialized first before the initialization of any instance variables.

Static Members

Methods declared as static (class methods) have several restrictions:

- They can only call other static methods.
- They must only access static data.
- They cannot refer to *this* or *super* in anyway.
- These methods can be accessed using the class name rather than a object reference.
- *main()* method should be always static because it must be accessible for an application to run, before any instantiation takes place.
- When *main()* begins, no objects are created, so if you have a member data, you must create an object to access it.

UML – Static data/method



```
public class Person {
        private String name;
        private int age;
        private static int personCounter;

        public Person(String name, int age) {
                ...
        } // Person

        public String getName() {
                ...
        } // getName

        public int getAge() {
                ...
        } // getAge

        public static int getPersonCounter() {
                ...
        } // getPersonCounter
} // Person
```

Static methods/Data members

```
public class Print  {
    public static String name = “default”;
    public static void printName()      {
            System.out.println(name);
    }
    public static void main(String arg[])  {
            System.out.println(Print.name);
            Print.printName();
    }
}
```

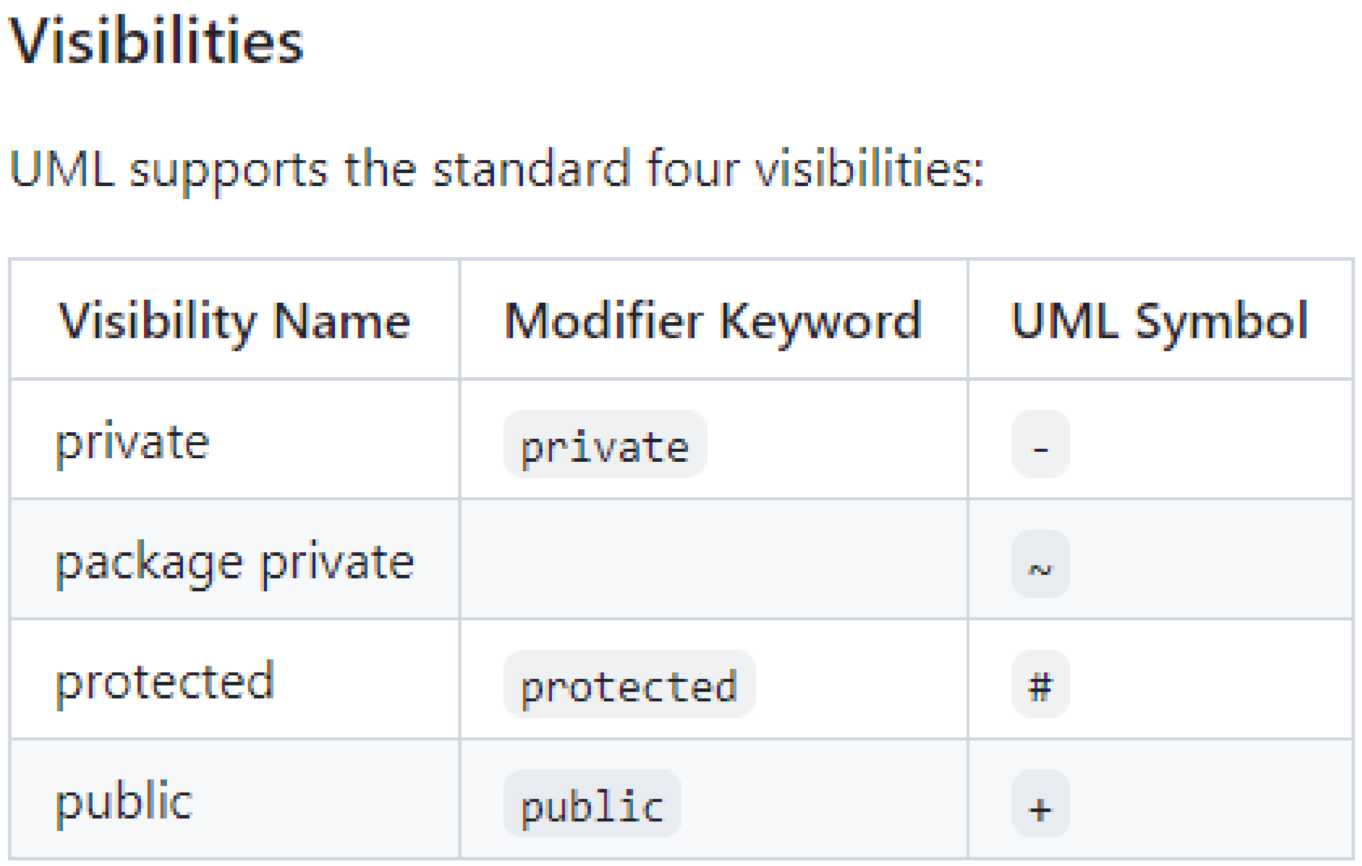
```
class TrackObj
{
        //class variable
        private static int counter = 0;

        //instance variable
        private int x = 0;

        TrackObj()
        {
                counter++;
                x ++;
        }

        //member method
        public int getX()
        {
                return x;
        }

        //class method
        public static int getCounter()
        {
                return counter;
        }
}
```

Access Modifiers

- Java provides a number of access modifiers to set the level of access for classes, fields, methods and constructors.
- A member has package or default accessibility when no accessibility modifier is specified.
- **Access Modifiers:**

  1. private 2. protected 3. default 4. public

Access Modifiers - UML



Visibilities

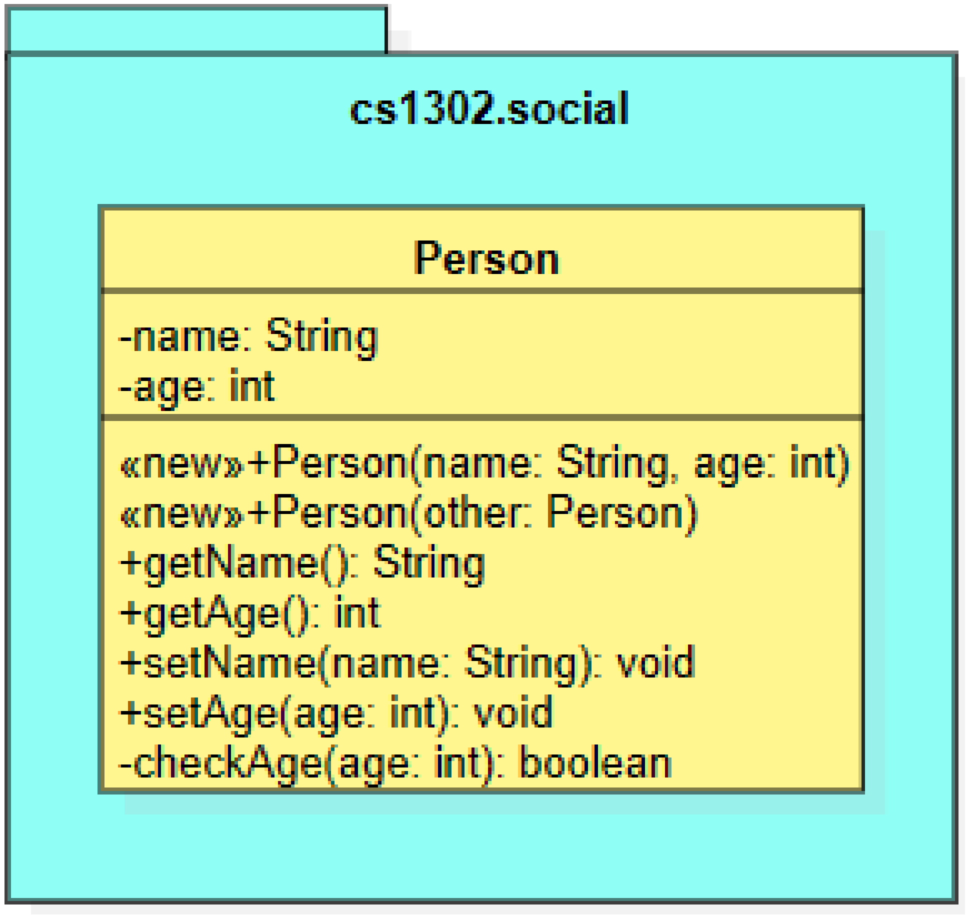
UML supports the standard four visibilities:

Visibility Name	Modifier Keyword	UML Symbol
private	`private`	`-`
package private		`~`
protected	`protected`	`#`
public	`public`	`+`

private access modifier

- The *private* (most restrictive) access modifier is used for fields or methods and cannot be used for classes and Interfaces.
- It also cannot be used for fields and methods within an interface.
- Field, method declared private are strictly controlled, and that member can be accessed only by other members of that class.
- A standard design strategy is to make all fields private and provide public getter methods for them.

private access modifier -UML



cs1302.social
Person
-name: String
-age: int
«new»+Person(name: String, age: int)
«new»+Person(other: Person)
+getName(): String
+getAge(): int
+setName(name: String): void
+setAge(age: int): void
-checkAge(age: int): boolean

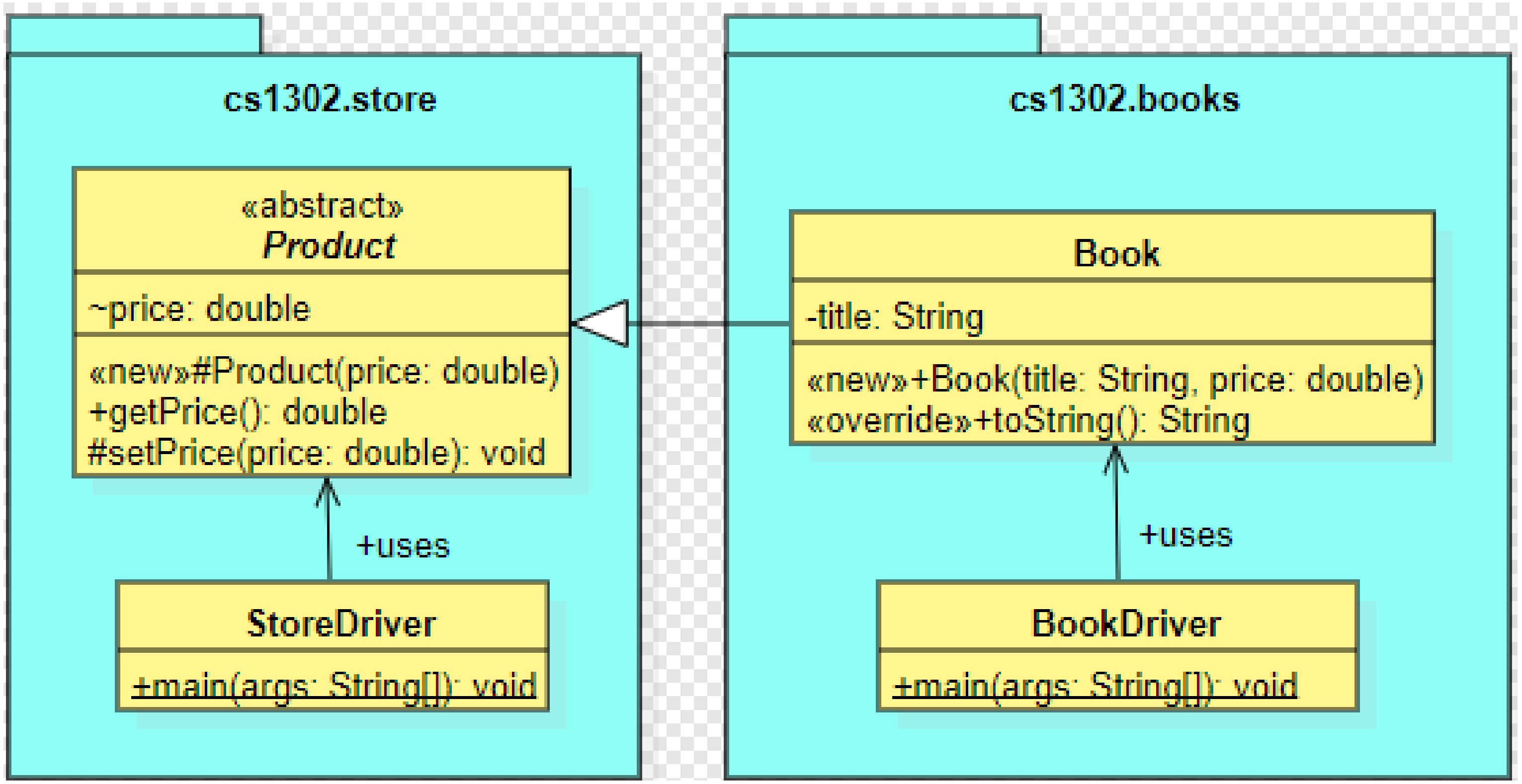
private access modifier -UML

```
// inside Person.java

public void setAge(int age) {

        if (!checkAge(age)) { // <---- LINE1 ✓
                throw new IllegalArgumentException("invalid age");
        } else {
                this.age = age; // <---- LINE2 ✓
        } // if

} // setAge
```

protected access modifier

- The *protected* access modifier is used for fields or methods and cannot be used for classes and Interfaces.
- It also cannot be used for fields and methods within an interface.
- Fields, methods and constructors declared protected in a superclass can be accessed only by its subclasses.
- Classes in the same package can also access protected fields, methods and constructors as well, even if they are not a subclass of the protected member's class.

protected access modifier

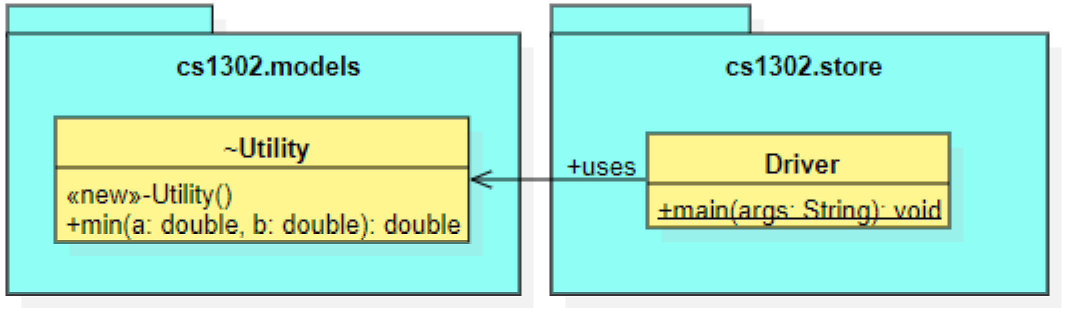

cs1302.store
«abstract»
Product
~price: double
«new»#Product(price: double)
+getPrice(): double
#setPrice(price: double): void
+uses
StoreDriver
+main(args: String[]): void
cs1302.books
Book
-title: String
«new»+Book(title: String, price: double)
«override»+toString(): String
+uses
BookDriver
+main(args: String[]): void

```
// inside Book.java (cs1302.books package) -- FIRST APPROACH
public Book(String title, double price) {
        super(price); // <---------------------------- LINE1
        this.title = title;
} // Book

// inside Book.java (cs1302.books package) -- SECOND APPROACH
public Book(String title, double price) {
        setPrice(price); // <------------------------- LINE2
        this.title = title;
} // Book

// inside BookDriver.java (cs1302.books package)
public static void main(String[] args) {
        Book lotr = new Book("The Lord of the Rings", 11.99);
        lotr.setPrice(lotr.getPrice() * 0.8); // <---- LINE3
} // main
```

default access modifier

- Java provides a default access modifier which is used when no access modifier is specified.
- Any class, field, method or constructor that has no declared access modifier is accessible only by classes in the same package.
- The default modifier is not used for fields and methods within an interface.

default access modifier - UML



```
// inside Utility.java
package cs1302.models;

class Utility { // <---- LINE1
    // ... rest omitted
```

```
// inside Driver.java
package cs1302.store;

import cs1302.models.Utility; // <---- LINE2

// ... rest omitted
```

public access modifier

- Fields, methods and constructors declared public (least restrictive) within a public class are visible to any class in the Java program, whether these classes are in the same package or in another package.

Access Levels

The following table shows the access to members permitted by each modifier.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N