Inheritance

zyBook Chap 10

Recap – Keyword `this`

Q: What is the meaning of the keyword 'this', and how can the keyword be used? Check ALL that apply.

- [x] It refers to the object on which a method or constructor has been called (sometimes called the "implicit parameter").
- [ ] It is used in conjunction with the 'that' keyword when the programmer wants to write a parallel "this and that" algorithm.
- [ ] It is used when one object wants to access data from a second object.
- [x] It can be used to access or set an object's field values
- [x] It can be used to call the object's methods.
- [ ] It is required when a class has more than one constructor.
- [x] It is used to call one constructor from another.

Recap – Superclass and Its Subclass(es)

- Subclass **extends** Superclass
  - Subclass is an instance of the superclass
    - Is-A relationship
  - Subclass can have new additional fields and methods
    - Extends/Increases the functionality of the superclass
    - **The superclass needs to be constructed first**
  - Within the subclass, a method that is inherited from the superclass can be overridden, e.g., toString(), equals()

ClientProgram.java | *Student.java | Person.java

```
1  import java.util.Arrays;
2
3  public class Student extends Person {
4
5      private String vuNetID;
6      private String[] courses;
7      private String[] grades;
8      private double gpa;
9
10     public Student() {
11         super();
12     }
13
14     public Student(int age, String name, String vuNetID) {
15         this.vuNetID = vuNetID;
16         super(age, name);
17     }
18
19     public String toStri
20         return super.toString() + " is a Student";
21     }
22
23 }
24
```

Constructor call must be the first statement in a constructor

Press 'F2' for focus

The **superclass** needs to be **constructed first**

Recap – Superclass and Its Subclass(es)

- **Constructors of superclass are NOT inherited**
  - Call it with the keyword `super`
- **Constructors in the subclass** need to
  - accept the same parameters as the superclass' constructor
  - in addition to parameters associated with specific data of the subclass

Keyword – `super`

- Provides a reference to the behavior of the superclass
  - Use to call **superclass constructor**
  - Use to call **superclass version of overridden methods**

Example Use of `super`

```java
public class Person {

    public int age;
    public String name;

    public Person() {
        this(0,"You-Know-Who");
     }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}
```

```java
public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return super.toString() + " is a Student";
    }
}
```

Call superclass constructor

Call superclass version of overridden methods

```java
public class Person {

    public int age;
    public String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return super.toString() + " is a Student";
    }
}
```

Q: What will be printed?

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

```
per P (age: 18)
stu S (age: 18) is a Student
```

```java
public class Person {

    public int age;
    public String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return name + " (age: " + age + ") is a Student";
    }
}
```

Q: How about this version?

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

```
per P (age: 18)
stu S (age: 18) is a Student
```

Q: How about this version?

```java
public class Person {

    public int age;
    public String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}
```

```java
public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return Person.toString() + " is a Student";
    }
}
```

Q: How about this version?

```java
public class Person {

    public int age;
    public String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}
```

ClientProgram.java | Student.java | Person.java

```java
1  import java.util.Arrays;
2
3  public class Student extends Person {
4
5      private String vuNetID;
6      private String[] courses;
7      private String[] grades;
8      private double gpa;
9
10     public Student() {
11         super();
12     }
13
14     public Student(int age, String name, String vuNetID) {
15         super(age, name);
16         this.vuNetID = vuNetID;
17     }
18
19     public String toString() {
20         return Person.toString() + " is a Student";
21     }
22
23 }
24
25
26
```

Cannot make a static reference to the non-static method toString() from the type Person

1 quick fix available:

Change 'toString()' to 'static'

```java
public class Person {

    private int age;
    private String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return name + " (age: " + age + ") is a Student";
    }
}
```

Q: What if the **age** and **name** are **private** fields in the Person Class? What will be printed?

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

Recap – Access Modifiers

Modifier	Description
public	Accessible by self, derived classes, and everyone else.
private	Accessible by self.

```java
public class Person {

    private int age;
    private String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return name + " (age: " + age + ") is a Student";
    }
}
```

Hint

Q: What if the **age** and **name** are **private** fields in the Person Class? What will be printed?

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

```
per P (age: 18)
Exception in thread "main" java.lang.Error:
Unresolved compilation problems:
The field Person.name is not visible
The field Person.age is not visible

at Student.toString(Student.java:19)
at ClientProgram.main(ClientProgram.java:10)
```

```java
public class Person {

    private int age;
    private String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    public String vuNetID;
    public String[] courses;
    public String[] grades;
    public double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return super.toString() + " is a Student";
    }
}
```

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

```
per P (age: 18)
stu S (age: 18) is a Student
```

Call the superclass version of toString() method.
The private fields are still visible to methods in Person.java

Access Modifier – protected

Modifier	Description
public	Accessible by self, **derived classes**, and everyone else.
private	Accessible by self.
protected	Accessible by self, **derived classes**, and other classes in the same package.
no specifier (default)	Accessible by self and other classes in the same package.

For better encapsulation, it is still preferred to set the superclass' fields as **private** and use **super**.getterMethod() in the subclasses

Demo of using protected

```java
public class Person {

    protected int age;
    protected String name;

    public Person() {
        this(0, "You-Know-Who");
    }

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return name + " (age: " + age + ")";
    }
}

public class Student extends Person {

    private String vuNetID;
    private String[] courses;
    private d String[] grades;
    protected double gpa;

    public Student() {
        super();
    }

    public Student(int age, String name, String vuNetID) {
        super(age, name);
        this.vuNetID = vuNetID;
    }

    public String toString() {
        return name + " (age: " + age + ") is a Student";
    }
}
```

```java
public class ClientProgram {
    public static void main(String[] args) {

        Person p = new Person(18, "per P");
        System.out.println(p.toString());

        Student s = new Student(18, "stu S", "sS");
        System.out.println(s.toString());
    }
}
```

```
per P (age: 18)
stu S (age: 18) is a Student
```

Override toString() and equals()

- toString()
- equals()

Recap (Lec32) – Override equals()

Compares two objects for **equality of some or all fields**

- Implemented within the class definition of the object under comparison
  - Compares the implicit parameter (current object) with the object passed as a parameter
  - That is, we need **the object to be compared** to be the **same object type** as the current object via
    - `instanceof` operator
    - Object casting

Recap (Lec32) – Override equals() in Book Class

```
public class Book {

    // ... Some instance variables and methods here ...

    public boolean equals(Object obj) {
        // The parameter obj could be any type of Object, such as a Point
        // Hence, we need to check if obj is an instance of Book Class
        if (obj instanceof Book) {
            // Object casting
            // Tell the compiler to treat the obj as if it is a Book object
            Book b = (Book) obj;
            // title and author are Strings, and hence compared with equals()
            // pubYear is an int, and hence compared with ==
            return title.equals(b.getTitle()) && author.equals(b.getAuthor())
                    && pubYear == b.getPubYear();
        } else {
            return false;
        }
    }
}
```

Override equals() in Person Class

```
public class Person {

    private int age;
    private String name;

    // ... Constructors & other methods ...

    public boolean equals(Object other) {
        // Check if other is an instance of Person
        if (other instanceof Person) {
            // Object casting
            // Tell the compiler to treat the Object other as if it is a Person object
            Person otherPerson = (Person) other;
            // age is an int, compared with ==
            // name is a String, compare with equals()
            return ____________________________________________________;
        } else {
            return false;
        }
    }
}
```

Override equals() in Person Class

```
public class Person {

    private int age;
    private String name;

    // ... Constructors & other methods ...

    public boolean equals(Object other) {
        // Check if other is an instance of Person
        if (other instanceof Person) {
            // Object casting
            // Tell the compiler to treat the Object other as if it is a Person object
            Person otherPerson = (Person) other;
            // age is an int, compared with ==
            // name is a String, compare with equals()
            return age == otherPerson.age && name.equals(otherPerson.name);
        } else {
            return false;
        }
    }
}
```

Override equals() in Student Class

```
public class Student extends Person {

    private String vuNetID;
    private String[] courses;
    private String[] grades;
    private double gpa;

    // ... Constructors & other methods ...

    public boolean equals(Object other) {
        // Check if other is an instance of Student
        if (other instanceof Student) {
            // Object casting
            Student otherStudent = (Student) other;
            // call the equals method from its superclass Person, which compares age and name
            // vuNetID is a String, compared with equals()
            return ____________________________________________________;
        } else {
            return false;
        }
    }
}
```

Override equals() in Student Class

```
public class Student extends Person {

    private String vuNetID;
    private String[] courses;
    private String[] grades;
    private double gpa;

    // ... Constructors & other methods ...

    public boolean equals(Object other) {
        // Check if other is an instance of Student
        if (other instanceof Student) {
            // Object casting
            Student otherStudent = (Student) other;
            // call the equals method from its superclass Person, which compares age and name
            // vuNetID is a String, compared with equals()
            return super.equals(otherStudent) && vuNetID.equals(otherStudent.vuNetID);
        } else {
            return false;
        }
    }
}
```