Java Inheritance

References

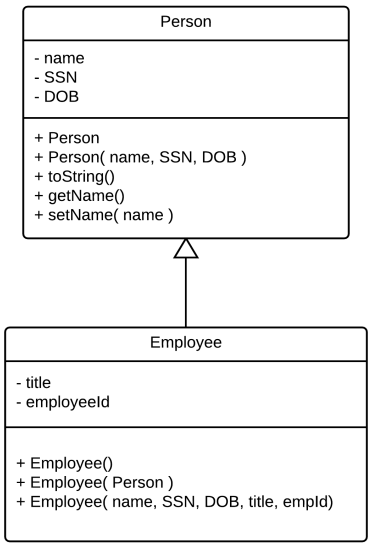
- tutorialspoint: *easy-to-understand*
  - Java Tutorial ← See section entitled Java Object Oriented
- Oracle Java doc: *more technical, but more complete*
  - Interfaces & Inheritance

Inheritance

- **Inheritance** is the process of *deriving* one class from another.
  - The **derived class (subclass)** gets all of the fields and methods of the other **base class (superclass)**.
  - "The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself." (oracle).

Inheritance UML

- Inheritance is used to implement an **IS-A** relationship.
- For example, an Employee is a Person.
- In the UML diagram, use an arrow pointing from subclass to superclass.
- You don't need to list inherited fields / methods in the subclass.



Superclass

- For example, consider our class Person:

```
public class Person {

	private String name;

	// initialize to default value, e.g. "anonymous"
	public Person(){}
	// initialize to parameter
	public Person(String n){ }

	// return value of _name
	public String getName(){ return name; }
	// change value of _name
	public void setName(String n){ name = n; }
}
```

Subclass

- We could create a subclass of person, for example, **Employee**
  - by extending, we automatically get all the fields and methods above.
  - we can add additional fields and methods.
  - when we override the constructor, we can call the constructor of the superclass using the **super** keyword.

```
public class Employee extends Person {

	int employee_id;
	String title;

	public Employee(int id) {
		employee_id = id;
		super();
	}

	public Employee(int id, String n) {
		employee_id = id;
		super(n);
	}

	 public int getId(){ return employee_id; }
}
```

Polymorphism

- **Polymorphism** is "the condition of occurring in several different forms."
    - In programming, it usually means using a superclass reference to refer to a subclass.
- For example, an Employee object can be stored in a Person collection.
    - *This is left as a programming exercise.*
- You can determine if a superclass object is an instance of a particular subclass by using the **instanceof** keyword.

instanceof keyword demo

```
Person p = new Employee( 0, "Jane" )
if( p instanceof Employee ){
    System.out.println( p.getName() + " is employee #" + p.getId() );
} else {
       System.out.println( p.getName() + " is unemployed." );
}
```

Reflection

- "Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine."
    - **Reflection** is a way for to get information about an object at runtime.
    - Reflection is an advanced topic, so we'll just scratch the surface:
- Java has an Object class and a Class class!
    - Every Java class has the **Object class** at the top of the hierarchy.
    - The **Class class** contains information about a class.
- It's best to see these methods in use:
    - Object::getClass() returns an object of type Class
    - Class::getSuperclass() returns superclass of a Class
    - Class::getDeclaredFields() gets all of the fields that are declared in that class (i.e. doesn't get fields of superclasses)

Reflection demo

```
import java.lang.reflect.Field;

public class InheritanceDemo {

	public static void main(String[] args){

		String s = "asdf";

		Class c = s.getClass();
		System.out.println("class:\n\t"+c);

		Class sup = c.getSuperclass();
		System.out.println("super:\n\t"+sup);

		System.out.println("fields:");
		Field[] fs = c.getDeclaredFields();
		for( int i=0; i<fs.length; i++){
			System.out.println("\t"+fs[i]);
		}
	}
}
```

Demo output

```
class:
	class java.lang.String
super:
	class java.lang.Object
fields:
	private final char[] java.lang.String.value
	private int java.lang.String.hash
	private static final long java.lang.String.serialVersionUID
	private static final java.io.ObjectStreamField[] java.lang.String.serialPersistentFields
	public static final java.util.Comparator java.lang.String.CASE_INSENSITIVE_ORDER
	private static final int java.lang.String.HASHING_SEED
	private transient int java.lang.String.hash32
```