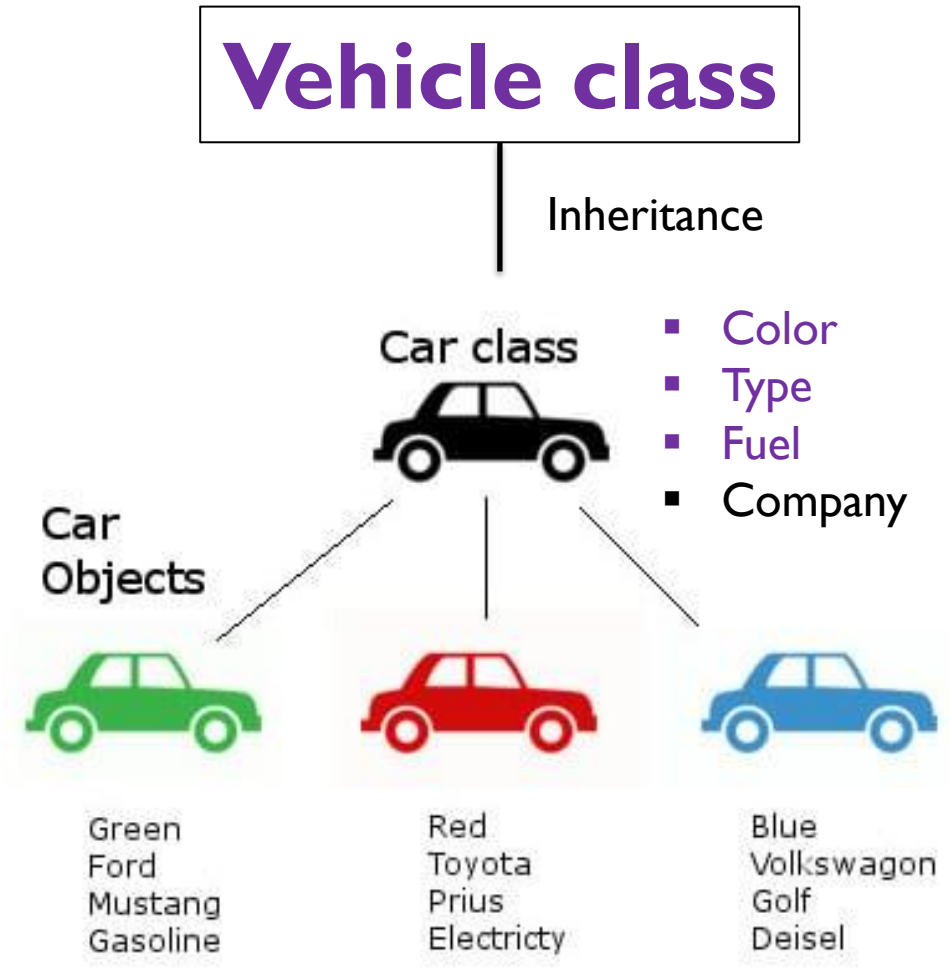
SSE3052: Embedded Systems Practice

Jinkyu Jeong
jinkyu@skku.edu
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Agenda

- Learning Java!
  - Installation
  - Class, method, and object
  - Inheritance
  - Predefined class example (String)

Object oriented programming



https://kienthuc24h.com/wp-content/uploads/2017/07/class.jpg

Installation

- JDK (Java Development Kit)
  - Check if JDK is installed (maybe not installed)
    - $javac –version
  - If not, install JDK
    - $sudo apt install default-jdk

- Eclipse IDE
  - Go to http://www.eclipse.org/downloads/
  - Download "Eclipse IDE for Java Developers"
  - Untar
  - Execute "eclipse-inst"

Compile and Execute (JDK)

- Compilation
  - $javac [java source code]
    - File extention of source code should be .java
    - Result of compile will be .class
    - If compile with other java source code, use –sourcepath option
- Execution
  - $java [class file]
- Simple example (Main.java)
  - $javac Main.java
    - Main.class will be created
  - $java Main

Warm Up (Hello World!)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

HelloWorld class

main method (~= function in C language)

public access modifier (public, private, protected)

C Code vs. Java Code

- C code

```
int funcA() {…}
int funcB() {…}
int funcC() {…}

int main() {
    funcA();
    funcB();
    funcC();
}
```

- Java code

```
class classA {
    int methodA() {…}
    int methodB() {…}
    int methodC() {…}
}
class classB {
    int methodA() {…}
    int methodB() {…}
    int methodC() {…}
}

class classC {
    static void main() {…}
}
```

Class

- A class is a template that describes data and behavior
- Similar with "structure" in C language

Ex)

```
class person {
    String firstName;              // Instance variable

    public String getFirstName() {      //Getter (method)
        return firstName;
    }

    public void setFirstName(String firstName) {      // Setter (method)
        this.firstName = firstName;
    }
}
```

```
class Person {
    String firstName;
    String lastName;
    int age;

    public void Person(String a, String b, int value) {
            firstName = a;
            lastName = b;
            age = value;
    }

    public String getFirstName() {
            return firstName;
    }

    public void setFirstName(String firstName) {
            this.firstName = firstName;
    }

    public String getLastName() {
            return lastName;
    }

    public void setLastName(String lastName) {
            this.lastName = lastName;
    }

    public int getAge() {
            return age;
    }

    public void setAge(int age) {
            this.age = age;
    }

    public String toString() {
            return firstName + " " + lastName;
    }
}
```

Constructor

Object

- An *object* is an instance of a *class*

Ex)

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person("Jim", "Knopf", 21);
        Person p2 = new Person("Jill", "Sanders", 20);
            // Jill gets married to Jim
            // and takes his name
        p2.setLastName("Knopf");
        System.out.println(p2);      // "Jill Knopf" will printed
    }
}
```

Constructor

- A special "method" that is invoked when creation of instance
- Name of constructor is ALWAYS the name of the class
- No return type
- One class can include several constructor (overloading)
- If no explicit constructor is defined, compiler implicitly adds a constructor with default values

Example (3 constructors)

```
class Person {
    String firstName;
    String lastName;
    int age;

    public Person() {

        firstName = "No";

        lastName = "Name";

        age = 0;

    }


    public Person(String a, String b) {

        firstName = a;

        lastName = b;

        age = 0;

    }


    public Person(String a, String b, int value) {
        firstName = a;
        lastName = b;
        age = value;
    }

    …
}
```

Method Overloading

- More than on method with the same name, different arguments
- Type of return value doesn't matter

Ex)

```
Class PrintStream {
    public void println(String s) {…}
    public void println(double a) {…}
}
```

(O)

```
class PrintStream {
    public void println(String s) {…}
    public int println(String s) {…}
}
```

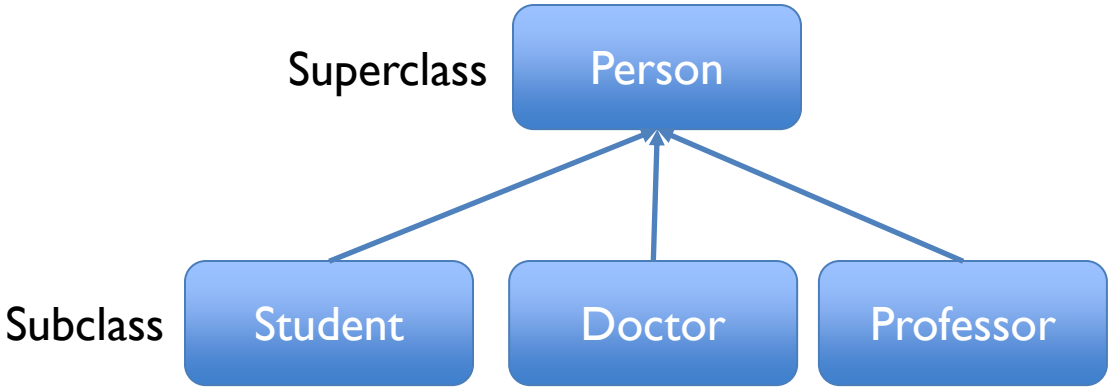
(X)

Exercise 1

- Implement a class *circle* that has methods *double getArea()* and *double getPerimeter()*. In the constructor, supply the radius of the circle. If radius is not given by user, default value is 10.

Inheritance

- A *subclass* which derived from *superclass*



Ex)

```
Class Person {
…
}

Class Students extends Person {
    int grade;

    public Student() {…}
}
```

Why Inheritance?

- Inheritance is a mechanism for
  - building class types from existing class types
  - defining new class types to be a
    - Specialization
    - Augmentation

    of existing types

Example (Student derived from Person)

```
class Student extends Person {
    int grade;

    public Student(String a, String b, int age, int grade) {
        super(a, b, age);     //Constructor of superclass Person
        this.grade = grade;
    }

    public int getGrade() {
        return grade;
    }

    public void setGrade(int grade) {
        this.grade = grade;
    }

    public String toString() {
        return getFirstName() + " " + getLastName() + " is grade " + grade;
    }
}
```

Example Cont. (Main)

```
class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Bill", "Gates", 13, 9);
        s1.setFirstName("Steve");
        s1.setGrade(10);
        System.out.println(s1);
        //”Steve Gates is grade 10” will printed
    }
}
```

Method Overriding

- Method redeclaration

Ex)

```
class BaseClass {
    public String toString() {
        return “Base”;
    }
}

class ChildClass extends BaseClass {
    public String toString() {
        return “Child”;
    }
}
```

Method Overriding

```
public class Main {
    public static void main(String[] args){
        BaseClass a = new BaseClass();
        ChildClass b = new ChildClass();
        System.out.println(a);          //"base"
        System.out.println(b);          //"child"
    }
}
```

String Class

- *String* class represents array of characters
  - *All string literals are implemented as instances of String class*

- Strings are immutable

- Initialization
  - Ex) String s = “text”;
  - Ex) String s = new String(“text”);

String Methods

- `char charAt(int index)`
  - Returns the char value at the specified index.
- `String concat(String str)`
  - Concatenates the specified string to be the end of this string.
- `boolean endsWith(String suffix)`
  - Tests if this string ends with the specified suffix.
- `boolean equals(Object anObject)`
  - Compares this string to the specified object.
- `int indexOf(int ch)`
  - Returns the index within this string of the first occurrence of the specified character.
- `int indexOf(String str)`
  - Returns the index within this string of the first occurrence of the specified substring.
- `int length()`
  - Returns the length of this string.
- More APIs: https://docs.oracle.com/javase/8/docs/api/

Exercise 2

- Write a superclass *Worker* and subclasses *HourlyWorker* and *SalariedWorker*. Every worker has a name and a salary rate. Write a method *computePay(int hours)* that computes the weekly pay for every worker(*override*). An hourly worker gets paid the hourly wage for the actual number of hours worked, if *hours* is at most 40. If the hourly worker worked more than 40 hours, the excess is paid at time and a half. The salaried worker gets paid the hourly wage for 40 hours, no matter what the actual number of hours is. Supply a test program to test these classes and methods. (Provide toString() for each classes)