09_1 Object Class

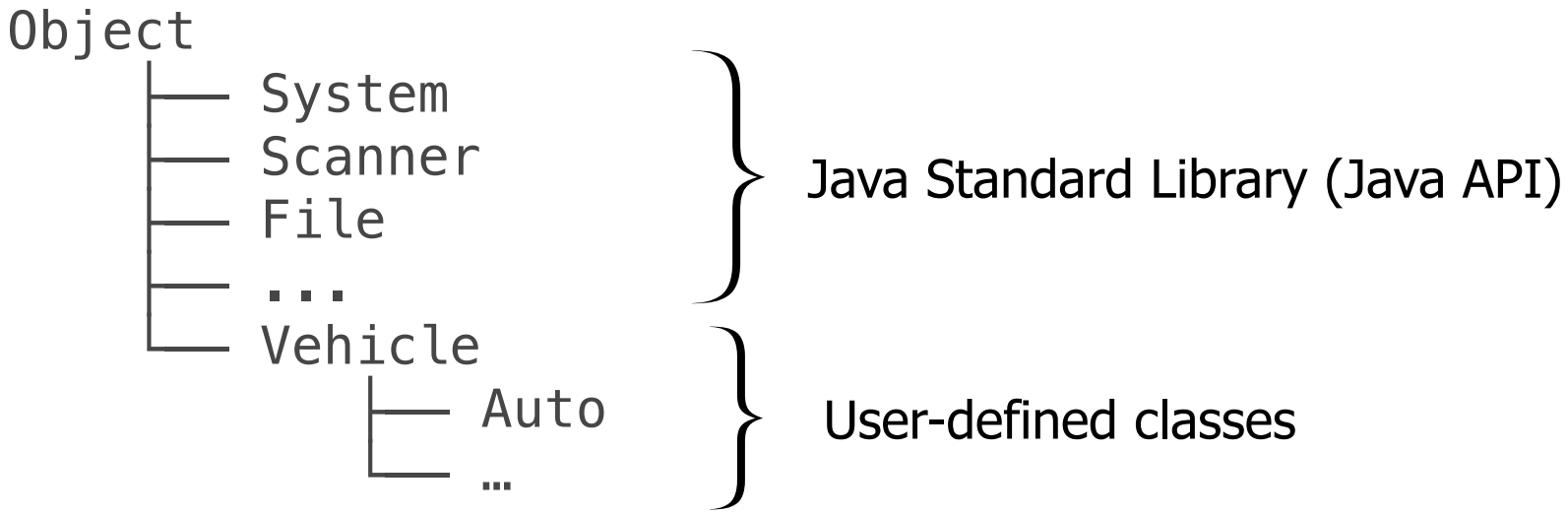
Object-Oriented Programming

java.lang package revisited

- **Object class**
- System class
- Class class
- String, StringBuilder, StringTokenizer
- Wrapper classes: Byte, Short, Character, Integer, Float, Double, Boolean
- Math class

Object class

- Inherited by default by all classes (in all packages) without 'extends' keyword
- Every class (including user-defined class) in java is a child or descendant of Object.



Methods in Object class (1/2)

- boolean equals(Object obj)
  - Testing whether two objects (caller and obj)’s references are the same or not
  - ‘equals(obj)’ is overridden and rewritten for almost all classes
- String toString()
  - Return the string representing the object’s information
  - toString() is overridden and rewritten for almost all classes

Methods in Object class (2/2)

- int hashCode()
  - Return the hash code (an integer identifying the object) of the object
  - The same objects should return the same hashCode()
  - The hashCode() method of Object class usually returns the memory address of the object
  - Overridden hashCode() usually implemented with the value of other instance variables
  - hashCode() is mainly used with hash-based collection such as ‘HashMap’, ‘HashSet’, and ‘HashTable’ (see later chapter for collections)
  - Multiplying prime number is a technique to reduce the hash collision

Example) ObjectClassTest (1/4)

```
class Person {
    private String name;
    private int age;

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
```

Example) ObjectClassTest (2/4)

```
    @Override
    public boolean equals(Object other) {

        if (this == other) return true;
        if (other == null || getClass() != other.getClass()) return false;

        Person person = (Person) other; // downcasting

        return age == person.age &&
           (name == null ?
            person.name == null :
            name.equals(person.name));

    }
```

Example) ObjectClassTest (3/4)

```
    @Override
    public int hashCode() {
        int result = ((name != null) ? name.hashCode() : 0);
        result = 31 * result + age; // multiplying the prime number 31
        return result;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Example) ObjectClassTest (4/4)

```
public class ObjectClassTest {

    public static void main(String[] args) {
        Person person1 = new Person("John", 25);
        Person person2 = new Person("John", 25);
        Person person3 = new Person("Jane", 30);

        System.out.println("person1 equals person2: "+person1.equals(person2)); // true
        System.out.println("person1 equals person3: "+person1.equals(person3)); // false

        System.out.println("person1 hashCode: " + person1.hashCode()); // 71750734
        System.out.println("person2 hashCode: " + person2.hashCode()); // 71850734
        System.out.println("person3 hashCode: " + person3.hashCode()); // 71339152

        System.out.println("person1 toString: " + person1.toString()); //...
        System.out.println("person2 toString: " + person2.toString());
        System.out.println("person3 toString: " + person3.toString());
    }
}
```

Example) EqualsWithPolymorphism (1/3)

```
class AClass {
    private int x;

    public AClass(int x) { this.x = x; }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (obj instanceof AClass) {
            AClass other = (AClass) obj;
            if (x == other.x) return true;
            return false;
        }
        return false;
    }
}
```

Polymorphism:
Descendant object can be assigned to Ancestor type reference variable

Example) EqualsWithPolymorphism (2/3)

```
class BClass extends AClass {
    private int y;
    public BClass(int x, int y) { super(x); this.y = y; }

    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (obj instanceof BClass) {
            BClass other = (BClass) obj; // downcasting
            if (super.equals(obj) && y == other.y)
                return true; // using super.equals
            return false;
        }
        return false;
    }
}

class CClass {
    private String name;
    public CClass(String name) { this.name = name; }
}
```

Example) EqualsWithPolymorphism (3/3)

```
public class EqualWithPolymorphism {
    public static void main(String[] args) {
        AClass a1 = new AClass(3);
        AClass a2 = new AClass(7);
        BClass b1 = new BClass(3, 5);
        BClass b2 = new BClass(7, 9);
        CClass c = new CClass("Korea");
        System.out.println(a1.equals(a1));  // true
        System.out.println(a1.equals(b1));  // true
        System.out.println(a1.equals(b2));  // false
        System.out.println(a2.equals(c));   // false
    }
}
```