JAVA

Lecture III – Object Oriented Programming

EXCERCISE

Which of the followings are valid identifiers?

a) _90ty

b) static

c) 目标

d) ABC

e) 12ab3

f) A_B_3_$

RECAP

- Identifiers:
  - $, _, [A-Za-z],[0-9].
  - Case-sensitive.
  - First character cannot be a digit.
  - Cannot be keywords.
  - No white space.
  - Camel-Case.
  - Classes, Interfaces, methods, variables, constants.

RECAP

- Primitive types:
  - Integers: byte, short, int, long
  - Floating-point: float, double
  - Characters: char
  - Boolean: boolean
- Operators: arithmetic, comparison, logical, assignment, conditional
- Control Statement:
  - If-Else,
  - switch,
  - Loops: for, while, do-while

ARRAYS

- Array: a collection of variables of the same type.
  - Fixed size.
  - Variables are of the same type.
  - One or more dimensions.
  - Implemented as objects. (unused arrays will be garbage collected)
- One-dimensional arrays

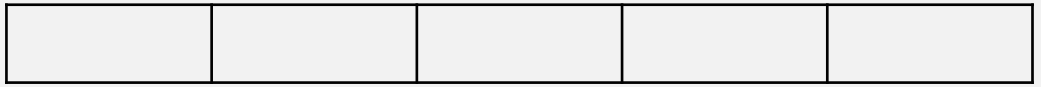
```
type[] array-name = new type[size];
```

- Type can be primitive types or any objects.

```
int[] intArray = new int[3];
String[] stringArray = new String[5];
```

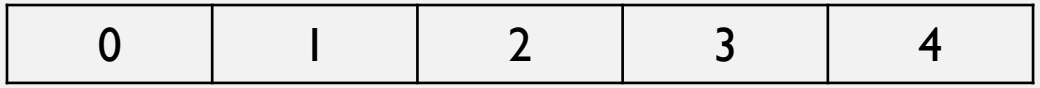
ARRAYS

- Each individual element within an array is accessed by its index.
- Index starts from zero, int[] xs = new int[5];



- Value assignement:
  - xs[3] = 5;
- Following code assign values to each element in an array

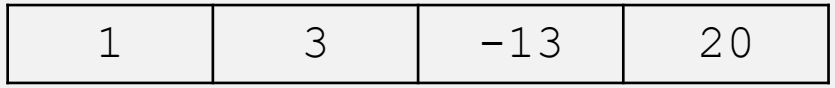
```
int[] xs = new int[5];
for(int i = 0; i < 5; i++)
        xs[i] = i;
```



ARRAYS

- Difficult to create an array and assign values like:

```
int[] xs = new int[4];
xs[0] = 1;
xs[1] = 3;
xs[2] = -13;
xs[3] = 20;
```



- Alternative, we could…

```
int[] xs = new int[]{1, 3, -13, 20};
int[] xs = {1, 3, -13, 20};
```

- Length of an array

```
xs.length;
```

EXAMPLE: MINMAX

Given a list of integer numbers nums (int[]), how to find out the maximum and minimum number in this list?

STRING[] IN MAIN

```
public static void main(String[] args)
```

`String[] args:` the input of the main method.

```
java FileName args…
```

```
public static void main(String[] args){
      System.out.println(args[0]); // print the first argument
}
```

```
java Test abc def
```

OBJECT-ORIENTED PROGRAMMING

- Procedural Programming: list of instructions to tell the computer what to do step by step. E.g., C

- Object-Oriented Programming (OOP): programs are organized around data. We define the data and the routines that are permitted to act on that data. E.g., Java, Python…

EXAMPLE

- Procedural Programming:
  - unlockDoor(door)
  - openDoor(door)
  - closeDoor(door)
  - lockDoor(door)
- OOP:
  - door.unlock()
  - door.open()
  - door.close()
  - door.lock()

CLASSES, OBJECTS AND METHODS

- Classes: **templates** that specify how to build individuals.
- Objects: **instances** of a particular class.

- Object-Oriented Programming: The data and the routines that are permitted to act on that data are grouped.

- “Code is organized around the data”

EXAMPLE: ELEPHANT

- Elephant (Class):
  - Data: name, color, height, weight, age, …
  - Methods: getColor(), getHeight(), MoveForward(), …

- An elephant named Bill (Object)
  - Data: specified values may be assigned to each data
  - Methods: all methods defined in its class can be called through this object

CLASS GENERAL FORM

```
class classname {
    // declare instance variables
    type varname;

    // declare constructors
    classname( parameters ) {
      // body of constructor
    }

    // declare methods
    type methodname( parameters ) {
      // body of method
    }
}
```

// Attributes of this class: different objects have different values.

// The method called when an object is initialized.

// Methods that act on the data

EXAMPLE: ELEPHANT

- A class with variables only.

```
class Elephant{
      String name;    // name of the elephant
      String color;   // color of the elephant
      int age;        // age of the elephant
}
```

- ElephantDemo: a class with main method to demonstrate

```
class ElephantDemo{
      public static void main(String[] args){
            Elephant bill;          // type declaration
            bill = new Elephant();
// create an object, "new" operator dynamically allocates memory
for this object and return a reference to it
      }
```

REFERENCE VARIABLES

- Object-Oriented Type Declaration:

```
Elephant bill, andy;
```

Creation of an object:

```
new Elephant();
```

- When we create an instance of class, the space is reserved in heap memory.

```
bill = new Elephant();
andy = new Elephant();
```

- The variable **bill** and **andy** refer to two different objects.
- Both are of the same form, but they have different copies of these 3 (instance) variables.
- To access instance variables, use the dot(.) operator:

```
object.member
```

- To change the `age` of **bill**:

```
bill.age = 5;
```

bill →

name	Bill
color	Grey
age	2

andy →

name	Andy
color	White
age	3

ASSIGNMENT OF PRIMITIVE TYPE

- `int x = 6; int y = 9;`
- If you assign

```
x = y;
```

- We change the value of y

```
y = 3;
```

- What is the value of x?

ASSIGNMENT OF PRIMITIVE TYPE

- `int x = 6; int y = 9;`
- If you assign

```
x = y;
```

- We change the value of y

```
y = 3;
```

- What is the value of y?

- `x = y;` //x receives a copy of the value of y

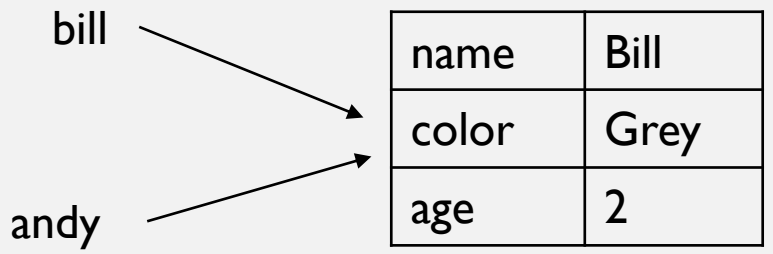
- What about assignment of reference type?

ASSIGNMENT OF REFERENCE VARIABLE

- If you assign

```
andy = bill;
```

- Then both variables refer to the same object



- If we change the value of `age` in andy, what is the age of bill?

```
andy.age = 5
```

- The object van previously referred to is garbage collected if there is no other
- reference to this object.

Garbage collected

- A safer (but less flexible) version of pointers.

EXAMPLE: VEHICLE

```
public class  Vehicle{
        int passengers;  // maximum number of passengers
        int fuelCap;     // fuel capacity in gallons
        int mpg;         // consumptions in miles per gallon
}
```

EXAMPLE: VEHICLE

```
public static void main(String[] args){
        Vehicle car = new Vehicle(); // create new objects
        Vehicle van = new Vehicle();
        car.mpg = 12;
        car.fuleCap = 14;
        van.mpg = 21;
        van.fuleCap = 16;
        car.mpg = van.mpg;   // what will happen?
        car.mpg = 10;        // what is the mpg value of van?
        van = car;           // what is the mpg value of car?
        van.mpg = 20;        // what is the mpg value of car?
}
```

EXAMPLE: VEHICLE

- We want to know the range of a vehicle:

```
public static void main(String[] args){
        Vehicle car = new Vehicle(); // create new objects
        Vehicle van = new Vehicle();
        car.mpg = 12;
        car.fuleCap = 14;
        van.mpg = 21;
        van.fuleCap = 16;
        int carRange = car.mpg * car.fuleCap;
        int vanRange = van.mpg * van.fuleCap;
}
```

- What is the problem here?

METHODS

- The user doesn't need to know how the range is calculated.
- Objects of the same class have similar ways to manipulate on its data.
- General Form

```
returnType methodName(parameters){
        statement;
        ……
}
```

- Parameters are local variables provided by the caller. Its scope is within the method.
- The return type can be any valid type of void.

EXAMPLE METHOD

- The following method can be added to the Vehicle class:

```
void range(){
        System.out.println("range: " + (fuleCap * mpg));
}
```

- If this method is called in the main method as follows:

```
        car.range();
```

- "range: 168" will be displayed
- Variables and Methods are accessed/called through the **object** not the class

RETURNING A VALUE

- return *value*;

```
int range(){
        return fuleCap * mpg;
}
```

- If the main method

```
int range = car.range();
System.out.println("range: " + range);
```

- Return type must be consistent with the definition.
- Return is the end of a method. Two ways to end a void method:
  - 1) reach the end brace }
  - 2) return;

PARAMETERS

- What if we want to know the amount of fuel needed to travel a certain distance?
- Parameters: local variable are provided by the caller whose scope is the method body.

```
double fuleNeeded(int distance){
        return (double)distance / mpg;
}
```

- In the main method:

```
System.out.println( car.fuleNeeded(120));
// print 10
```

CONSTRUCTORS

- Initial variable values are set in the main method.
  - Difficult to manage
  - Error Prone
- Constructor is used to initialize an object when it is created.

```
Vehicle(){
        fuleCap = 14;
        mpg = 12;
        passages = 4;
}
```

- The name of a constructor must be consistent with its class name

```
Vehicle car = new Vehicle();
```

CONSTRUCTORS

- Problem: different objects may have different initial values.
- Constructor with parameters

```
Vehicle(int f, int m, in p){
        fuleCap = f;
        mpg = m;
        passages = p;
}
```

- In the main method:

```
Vehicle car = new Vehicle(14, 12, 4);
```

- Variables can be initialized in the constructor or in the variable declaration, what is the order?

EXAMPLE

```
class ThisTest{
	int a, b;
	public void setData(int a, int b){
		a = a;
		b = b;
	}
	public void showData(){
		System.out.println("a=" + a);
		System.out.println("b=" + b);
	}
	public static void main(String[] args){
		ThisTest tt = new ThisTest();
		tt.setData(10, 20);
		tt.showData();
	}
}
```

- What is the output?

KEYWORD *THIS*

- Problem: the value of a local variable is assigned to an instance variable of the same name.
- ***this***: an implicit argument that refers to the object on which the method is called.
- You don't have to create an object. this help you to clarify instance variables and local variables.

```
public void setData(int a, int b){
      this.a = a;
      this.b = b;
}
```

EXAMPLE

```
class ThisTest{
	int a, b;
	public void setData(int a, int b){
		this.a = a;
		this.b = b;
	}
	public void showData(){
		System.out.println("a=" + a);
		System.out.println("b=" + b);
	}
	public static void main(String[] args){
		ThisTest tt = new ThisTest();
		tt.setData(10, 20);
		tt.showData();
	}
}
```

- What is the output?

DESIGN PATTERN - BUILDER

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

    private Person(Builder builder) {
        this.name = builder.name;
        this.age = builder.age;
}
```

DESIGN PATTERN - BUILDER

```
public static class Builder {
        private String name;
        private int age;

        public Builder name(String name) {
            this.name = name;
            return this;
        }
        public Builder age(int age) {
            this.age = age;
            return this;
        }
        public Person build() {
            return new Person(this);
        }
    }
    public void printDetails() {
        System.out.println("Name: " + name + ", Age: " + age + ");
    }
}
```

DESIGN PATTERN - BUILDER

```
Person person = new Person.Builder()
                    .name("John")
                    .age(30)
                    .build();

person.printDetails();
```