Tutorial

Bran & Yvonne

Quiz 1

1. What is the course about?

a. Programming in a language

b. How to compile a program

c. The design and implementation of languages

d. How to interpret a program

2. Which one of following is not a programming paradigm?

a. Statement

b. Object-oriented

c. Functional

d. Declarative

3. Which one is not a basic property of programming languages?

a. Syntax
b. Names
c. Types
d. Values

Syntax, names, types, semantics

4. What is not a major concern of syntax?

a. Syntax error detection

b. Types

c. Grammar

d. Vocabulary

5. A name (identifier) always refers to the same identity, regardless of the context(scope, visibility, lifetime, type...)

a. True

b. False

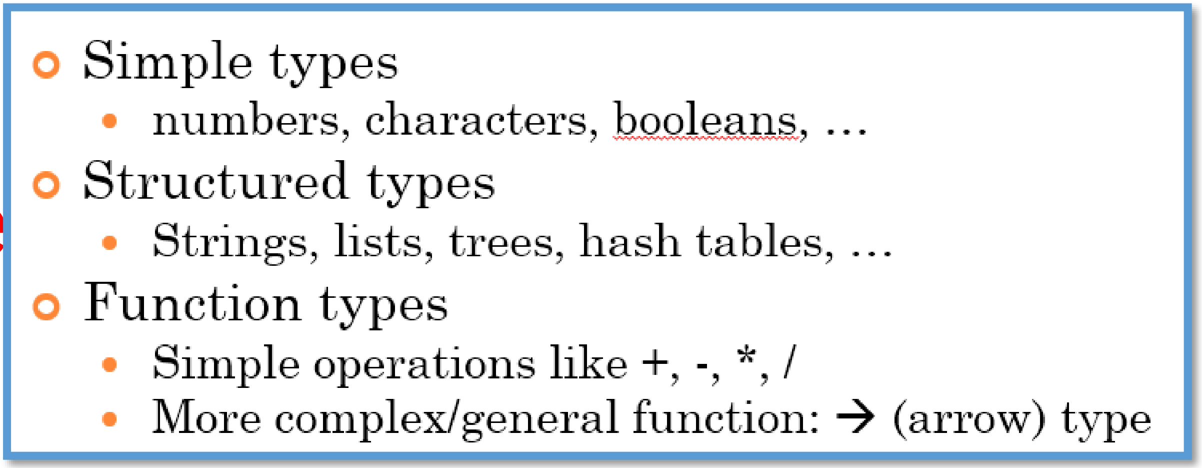
6. Which of the following is not a type in imperative programs?

a. Arrow (function) type

b. Boolean

c. String

d. Integer

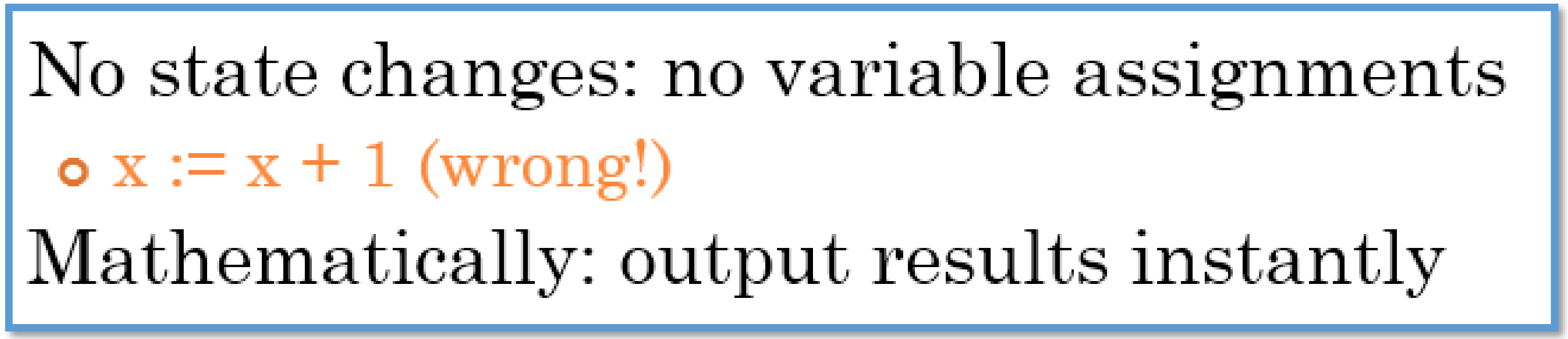


- Simple types
  - numbers, characters, booleans, …
- Structured types
  - Strings, lists, trees, hash tables, …
- Function types
  - Simple operations like +, -, *, /
  - More complex/general function: → (arrow) type

7. Is state change a major feature in function language?

a. Yes

b. No



No state changes: no variable assignments

- $x := x + 1$ (wrong!)

Mathematically: output results instantly

8. Logic programs describes?

a. How the problem is solved

b. What is the expected outcome

9. What doesn’t make a successful programming language?

a. Clarity about binding
b. Abstraction
c. Orthogonality
d. Similarity with human language

10. Which language is Python?

a. A compiled language: C/C++

b. An interpreted language: Python

c. A hybrid compilation/interpretation: Java

d. None of the above

Homework1

Problem 1. (30 points) Give a feature of C, C++ or Java that illustrates orthogonality. Give a feature that illustrates non-orthogonality.

- can be used independently, without affecting each other
- no exception
- No redundancy
- No overlap

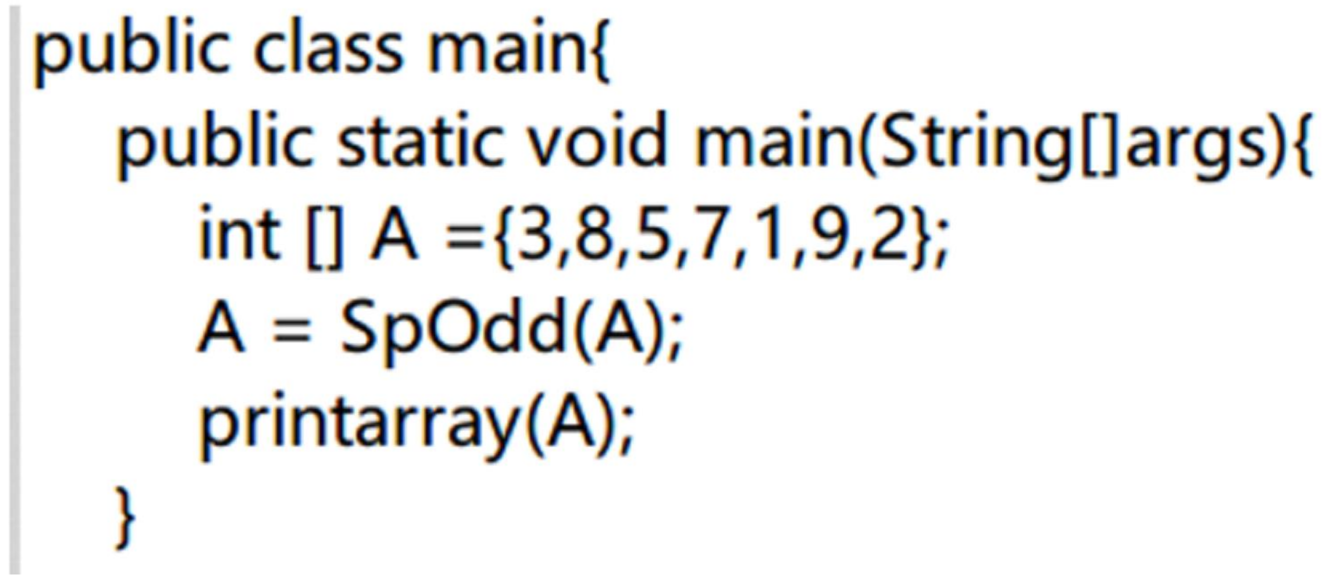
Homework1: Problem3

Problem 3. (40 points) We have learned the difference between compiler and interpreter. Now research compiled languages and interpreted languages. Then list the advantages and disadvantages of these two languages.

- Runtime speed?
- Time for compilation?
- Cross-platform?
- Program size?
- Easy to debug?

Homework1

- method name is the same as the main class

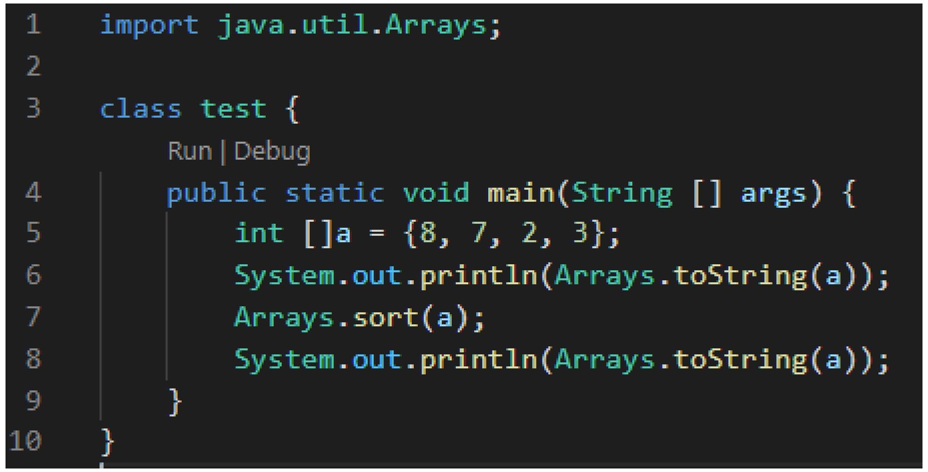


```
public class main{
    public static void main(String[]args){
        int [] A ={3,8,5,7,1,9,2};
        A = SpOdd(A);
        printarray(A);
    }
```

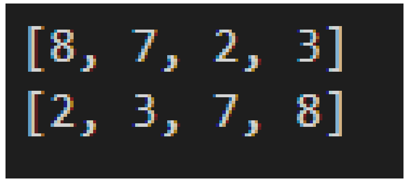
- The name of .java file must be the same as the class with main function so that the compiler knows where to start.

Homework1

- Print arrays & sort arrays



```java
import java.util.Arrays;

class test {
    Run | Debug
    public static void main(String [] args) {
        int []a = {8, 7, 2, 3};
        System.out.println(Arrays.toString(a));
        Arrays.sort(a);
        System.out.println(Arrays.toString(a));
    }
}
```

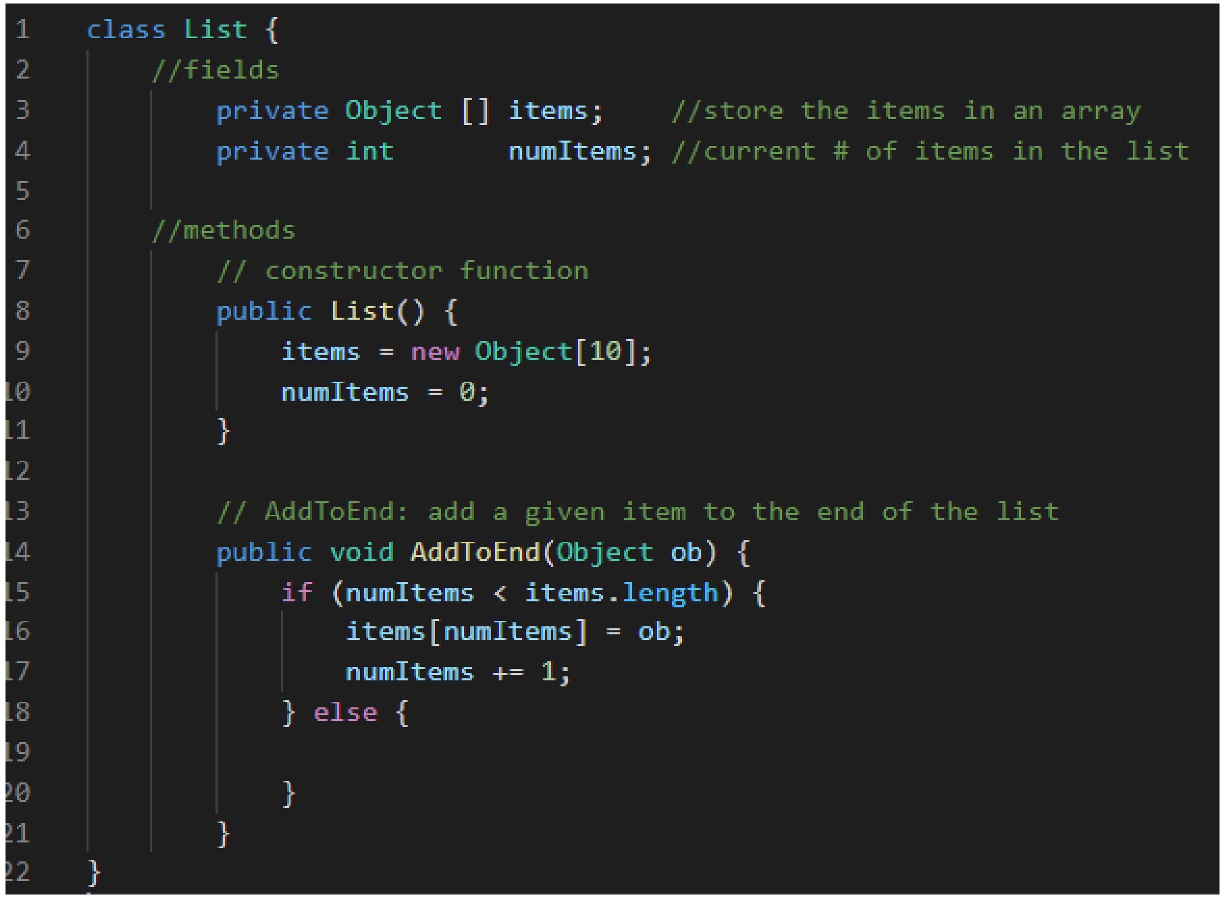


```
[8, 7, 2, 3]
[2, 3, 7, 8]
```

Java classes

Fields and methods

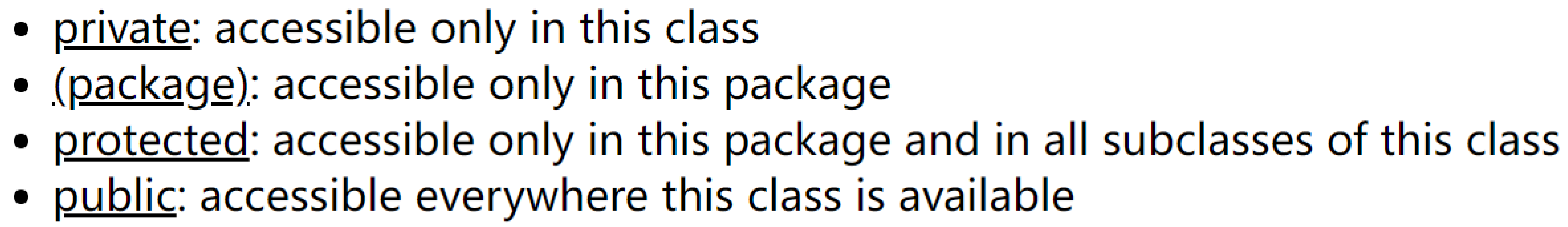
Java classes contain *fields* and *methods*. A field is like a C++ data member, and a method is like a C++ member function.



```java
class List {
    //fields
        private Object [] items;    //store the items in an array
        private int       numItems; //current # of items in the list

    //methods
        // constructor function
        public List() {
            items = new Object[10];
            numItems = 0;
        }

        // AddToEnd: add a given item to the end of the list
        public void AddToEnd(Object ob) {
            if (numItems < items.length) {
                items[numItems] = ob;
                numItems += 1;
            } else {

            }
        }
}
```

Access level

Each field and method has an *access level*



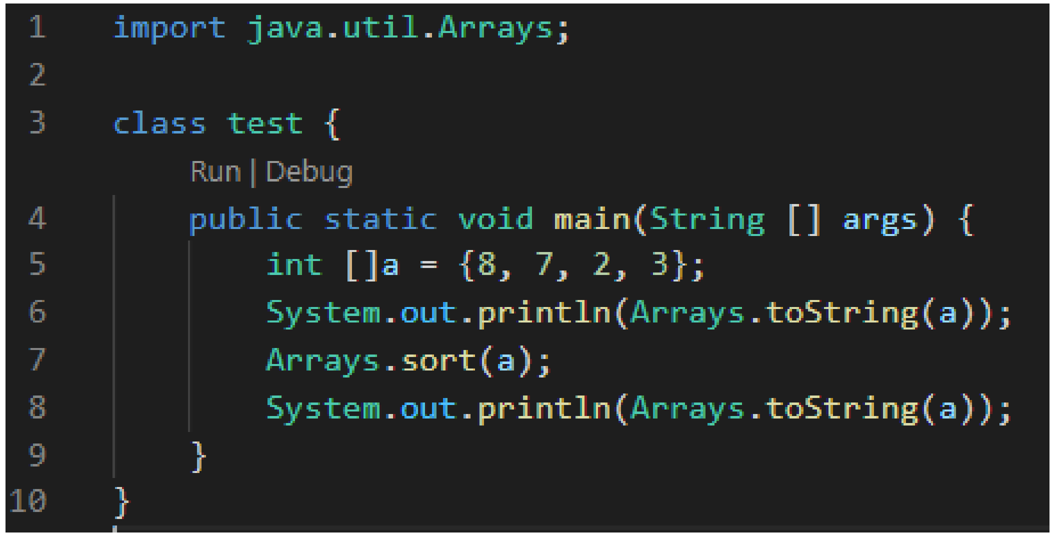
- private: accessible only in this class
- (package): accessible only in this package
- protected: accessible only in this package and in all subclasses of this class
- public: accessible everywhere this class is available

Access level

Similarly, each class has one of two possible access levels:

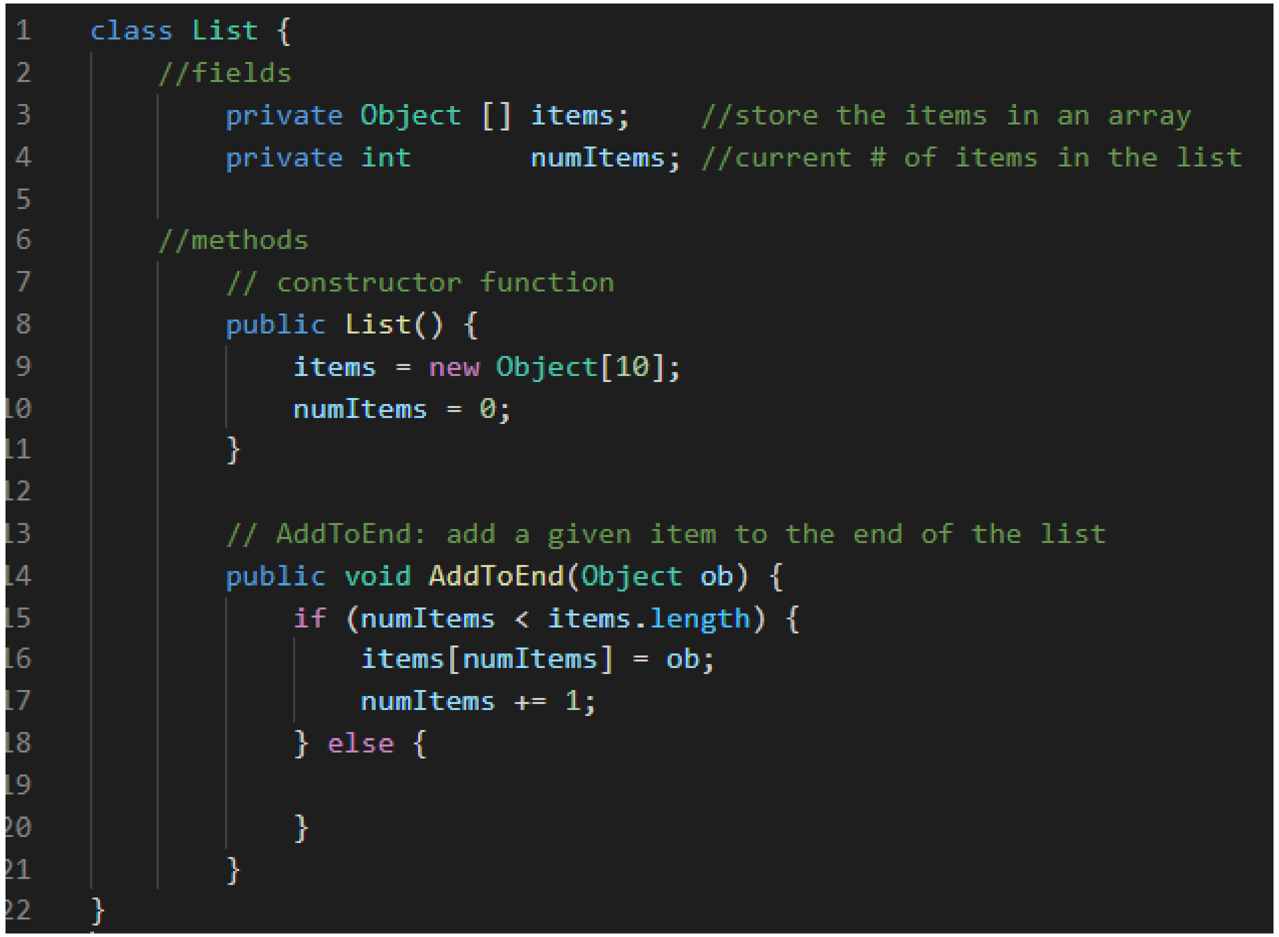
- (package): class objects can only be declared and manipulated by code in this package
- public: class objects can be declared and manipulated by code in any package

For both fields and classes, package access is the default, and is used when *no* access is specified.



```java
import java.util.Arrays;

class test {
    Run | Debug
    public static void main(String [] args) {
        int []a = {8, 7, 2, 3};
        System.out.println(Arrays.toString(a));
        Arrays.sort(a);
        System.out.println(Arrays.toString(a));
    }
}
```

Access level



```
class List {
    //fields
        private Object [] items;    //store the items in an array
        private int       numItems; //current # of items in the list

    //methods
        // constructor function
        public List() {
            items = new Object[10];
            numItems = 0;
        }

        // AddToEnd: add a given item to the end of the list
        public void AddToEnd(Object ob) {
            if (numItems < items.length) {
                items[numItems] = ob;
                numItems += 1;
            } else {

            }
        }
}
```

Object

Object-oriented programming involves *inheritance*. In Java, all classes (built-in or user-defined) are (implicitly) subclasses of Object. Using an array of Object in the List class allows any kind of Object (an instance of any class) to be stored in the list. However, primitive types (int, char, etc) cannot be stored in the list.

Constructor function

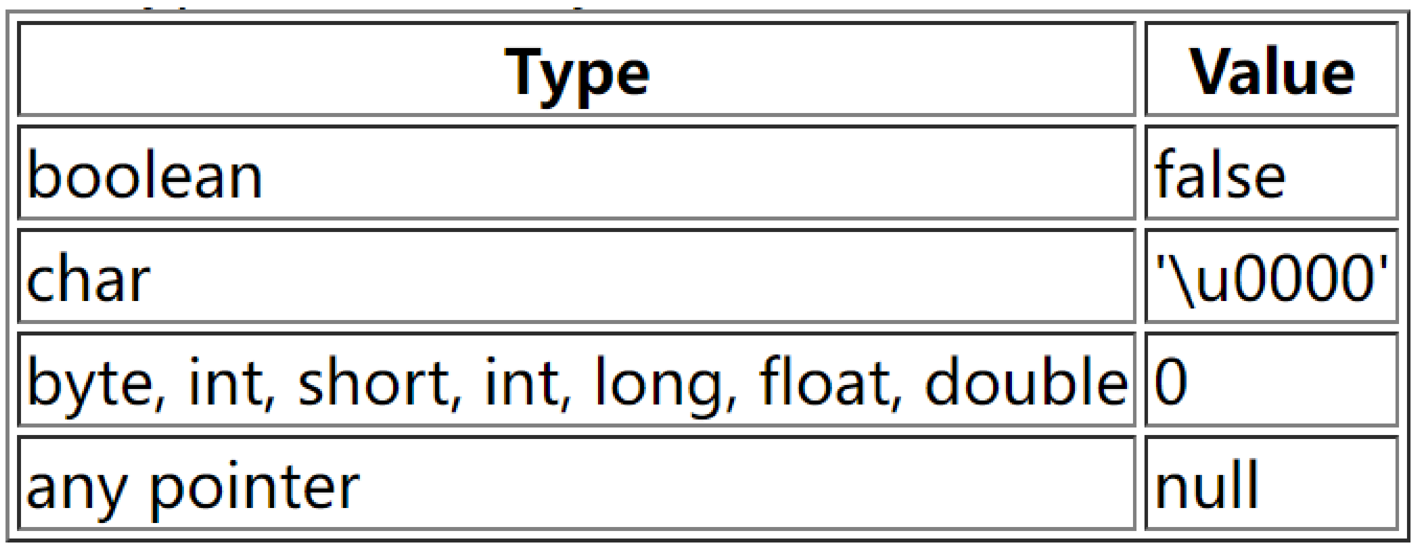
- Are used to initialize each instance of a class.
- Have no return type (not even void).
- Can be overloaded; you can have multiple constructor functions, each with different numbers and/or types of arguments.

If you don't write any constructor functions, a default (no-argument) constructor (that doesn't do anything) will be supplied.

If you write a constructor that takes one or more arguments, no default constructor will be supplied (so an attempt to create a new object without passing any arguments will cause a compile-time error).

Initialization of fields

If you don't initialize a field (i.e., either you don't write any constructor function, or your constructor function just doesn't assign a value to that field), the field will be given a default value, depending on its type. The values are the same as those used to initialize newly created arrays

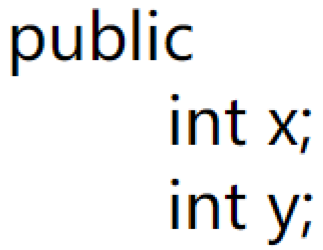


Type	Value
boolean	false
char	'\u0000'
byte, int, short, int, long, float, double	0
any pointer	null

Access Control

Note that the access control must be specified for every field and every method; there is no grouping as in C++.

For example, given these declarations:



```
public
        int x;
        int y;
```

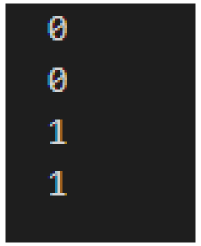
only x is public; y gets the default, package access.

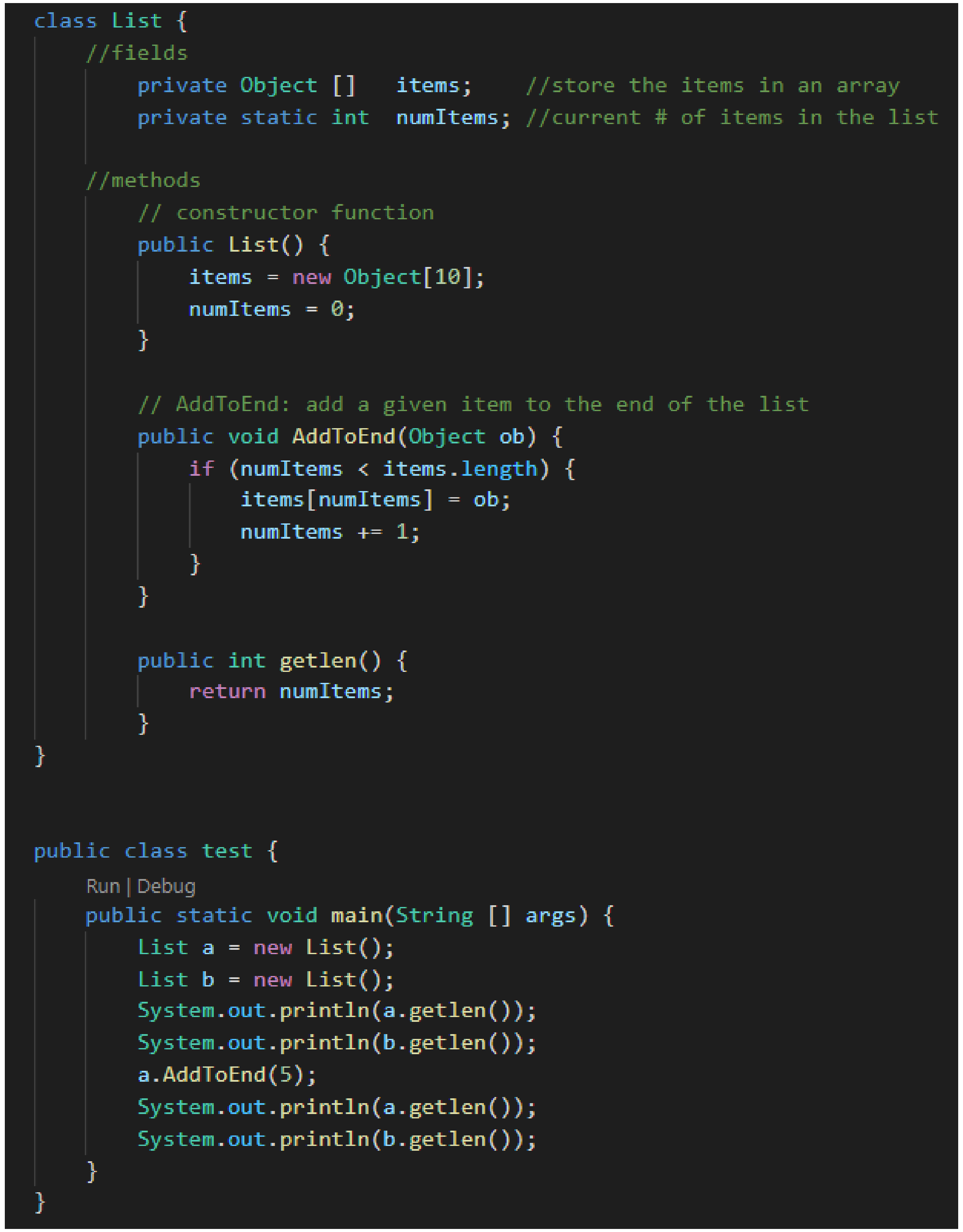
Static Fields and Methods

Fields and methods can be declared *static*

If a field is static, there is only one copy for the entire class, rather than one copy for each instance of the class.

Static Fields and Methods


```
0
0
1
1
```


```
class List {
    //fields
        private Object []   items;    //store the items in an array
        private static int  numItems; //current # of items in the list

    //methods
        // constructor function
        public List() {
            items = new Object[10];
            numItems = 0;
        }

        // AddToEnd: add a given item to the end of the list
        public void AddToEnd(Object ob) {
            if (numItems < items.length) {
                items[numItems] = ob;
                numItems += 1;
            }
        }

        public int getlen() {
            return numItems;
        }
}


public class test {
    Run | Debug
    public static void main(String [] args) {
        List a = new List();
        List b = new List();
        System.out.println(a.getlen());
        System.out.println(b.getlen());
        a.AddToEnd(5);
        System.out.println(a.getlen());
        System.out.println(b.getlen());
    }
}
```

Static Fields and Methods

A method should be made static when it does not access any of the non-static fields of the class, and does not call any non-static methods. (In fact, a static method *cannot* access non-static fields or call non-static methods.)

Methods which only manipulate the class's static fields should be static.

For example, if we wanted a function to print the current value of the numItmes field defined above, that function should be defined as a static method of the List class.
(getlen())

Static Fields and Methods

A public static field or method can be accessed from outside the class using either the usual notation:

```
class-object.field-or-method-name
```

or using the class name instead of the name of the class object:

```
class-name.field-or-method-name
```

The *preferred* way to access a static field or a static method is using the class name (not using a class object). This is because it makes it clear that the field or method being accessed is static.

Final Fields and Methods

Fields and methods can also be declared *final*. A final method cannot be overridden in a subclass. A final field is like a constant: once it has been given a value, it cannot be assigned to again.

Test 1

Consider the program defined below.

```
class Test {
  static int x;
  int k;

  // constructor with 2 args
  public Test( int n, int m ) {
    x = n;
    k = m;
  }

  public static void main(String[] args) {
    Test t1 = new Test(10, 20);
    Test t2 = new Test(30, 40);
    System.out.print(t1.x + " ");
    System.out.print(t1.k + " ");
    System.out.print(t2.x + " ");
    System.out.println(t2.k);
  }
}
```

Question 1: Which of the following is true?

A. This program must be in a file called Test.java. Compiling will create one new file called Test.class.
B. This program can be in any .java file. Compiling will create one new file called Test.class.
C. This program must be in a file called Test.java. Compiling will create two new files called Test.class and main.class.
D. This program can be in any .java file. Compiling will create two new files called Test.class and main.class.

Question 2: Which of the following correctly describes what happens when the program is compiled and run?

A. There will be a compile-time error because there is no constructor with no arguments.
B. There will be a run-time error because there is no constructor with no arguments.
C. There will be no errors; the output will be: 10 20 30 40
D. There will be no errors; the output will be: 30 20 30 40
E. There will be no errors; the output will be: 30 40 30 40

Question 1:

B. This program can be in any .java file. Compiling will create one new file called Test.class. (It does not have to be in Test.java, because class Test is not public. No file ″main.class″ is created, because main is a method, not a class.)

Question 2:

D. There will be no errors; the output will be: 30 20 30 40
(There is no need for a constructor with no arguments because there are no uses of ″new Test″ with no arguments.)