Chapter 4.

Revision on Java and OOP

Note:
This set of slides is for students who do not know Java or even programming. If you are Java expert, you can skip it.

2023-2024
COMP7506 Smart Phone Apps Development
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

Agenda

- Running Java
- Revision on Java Syntax
- OOP Concept & Its Importance in Mobile Apps Development

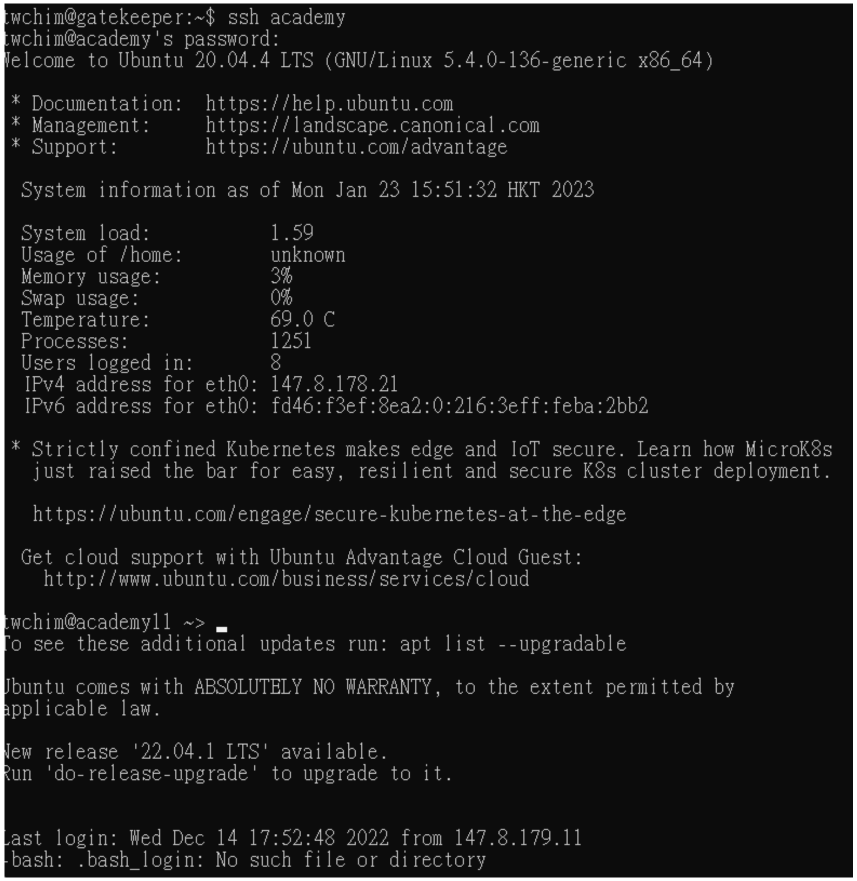
Running Java

Getting and using Java

- JDK can be freely downloaded from: https://www.oracle.com/java/technologies/javase-downloads.html
  - Latest version: JDK 21
  - May need to set path for Windows
    - E.g., C:\Program Files (x86)\Common Files\Oracle\Java\javapath
    - More information: http://java.com/en/download/help/path.xml)
- All text editors support java
  - Vi / Vim, Emacs, NotePad, WordPad
  - Just save as *.java file
- Eclipse IDE
  - Eclipse: https://www.eclipse.org/downloads/packages/
  - Android Development Tools (ADT) is a plugin for Eclipse
  - In our course, we will use Android Studio (i.e., one package contains everything we need for Android apps development).
    - http://developer.android.com/sdk/index.html
- Note: JDK is required for running Android Studio but Eclipse is not.

Running Java on academy.cs.hku.hk

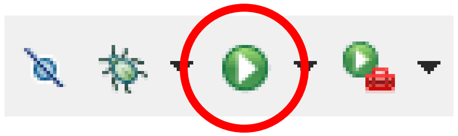
- If you don’t want to install JDK and Eclipse, you can also run Java programs using any online compiler.
- You can also use our Ubuntu server “academy11.cs.hku.hk”.
- To connect to it, you can “ssh” to it via the gateway “gatekeeper.cs.hku.hk”.
  - Open a Windows Command / Mac Terminal window.
  - Issue the command “ssh <CS_username>@gatekeeper.cs.hku.hk” and type your CS password.
  - Then issue the command “ssh academy11” and type your CS password again.



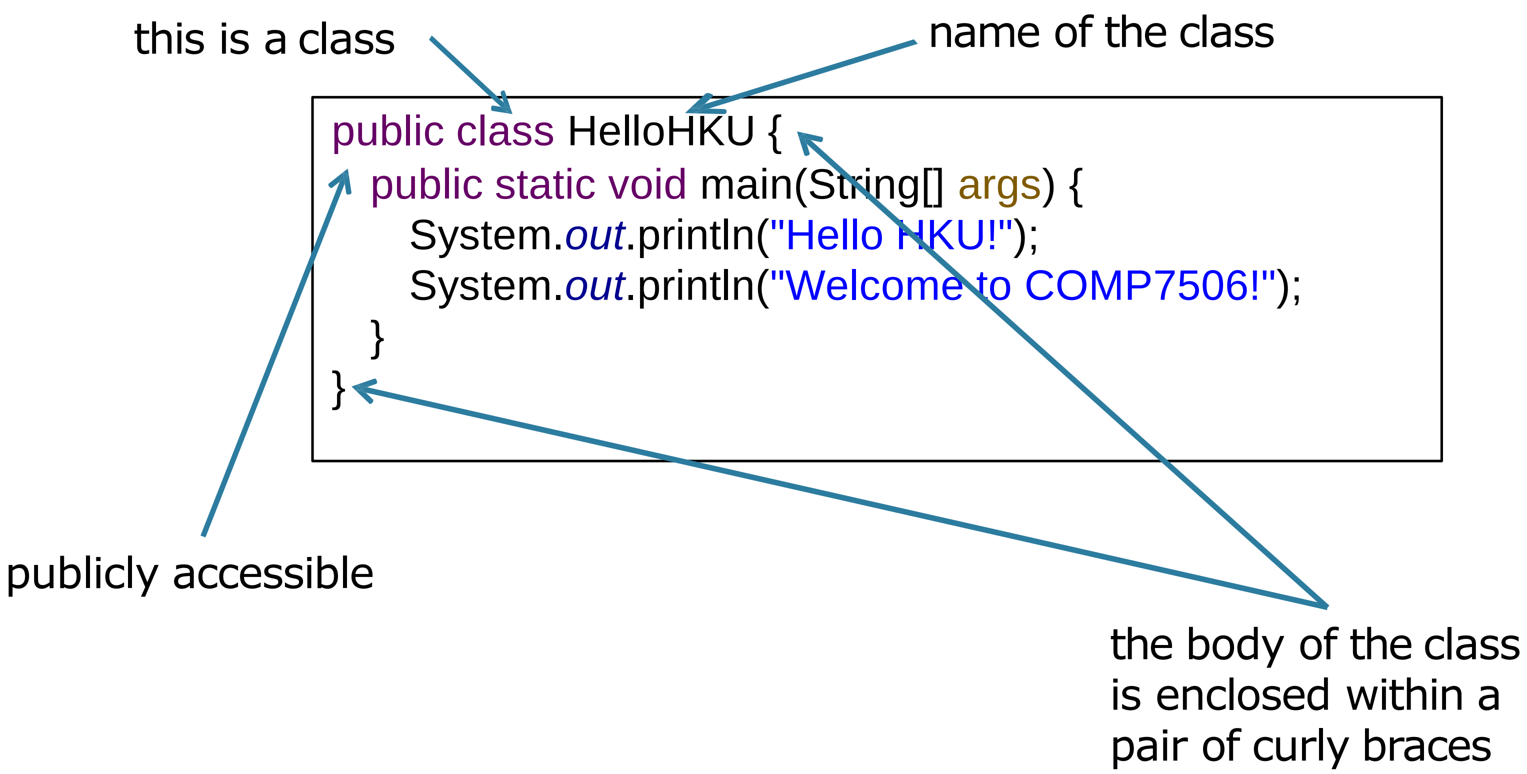
```
twchim@gatekeeper:~$ ssh academy
twchim@academy's password:
elcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-136-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Jan 23 15:51:32 HKT 2023

  System load:           1.59
  Usage of /home:        unknown
  Memory usage:          3%
  Swap usage:            0%
  Temperature:           69.0 C
  Processes:             1251
  Users logged in:       8
  IPv4 address for eth0: 147.8.178.21
  IPv6 address for eth0: fd46:f3ef:8ea2:0:216:3eff:feba:2bb2

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

twchim@academy11 ~> _
o see these additional updates run: apt list --upgradable

buntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
pplicable law.

ew release '22.04.1 LTS' available.
un 'do-release-upgrade' to upgrade to it.


ast login: Wed Dec 14 17:52:48 2022 from 147.8.179.11
bash: .bash_login: No such file or directory
```

Compile and run an application

- Write java class HelloHKU containing a main() method and save in file “HelloHKU.java”
    - The file name MUST be the same as class name
- If you are using command line environment (“cmd” in Windows) or Linux environment:
    - Compile with: javac HelloHKU.java
    - Creates compiled .class file: HelloHKU.class
    - Run the program: java HelloHKU
        - Notice: use the class name directly, no .class!
- If you are using IDE like Eclipse:
    - Compile and run by pressing the arrow button.



“HelloHKU” Example

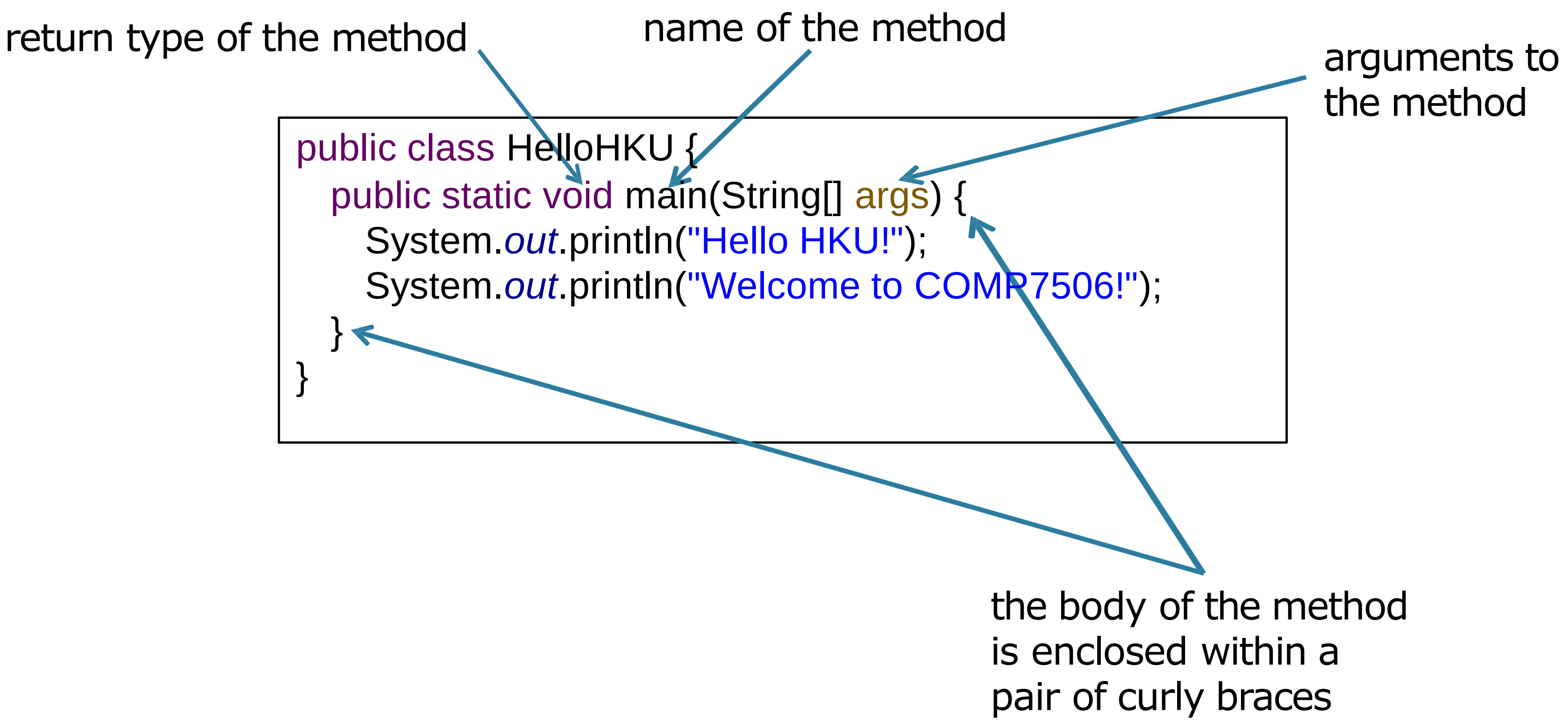


```
public class HelloHKU {
  public static void main(String[] args) {
    System.out.println("Hello HKU!");
    System.out.println("Welcome to COMP7506!");
  }
}
```

this is a class
name of the class
publicly accessible
the body of the class is enclosed within a pair of curly braces

- Every Java application has to have at least one class
- The main Java class (that is the class you call using “java”) has to have one main() method
- The source code should be saved in a file named HelloHKU.java

“HelloHKU” Example



```
public class HelloHKU {
   public static void main(String[] args) {
      System.out.println("Hello HKU!");
      System.out.println("Welcome to COMP7506!");
   }
}
```

return type of the method
name of the method
arguments to the method
the body of the method is enclosed within a pair of curly braces

“HelloHKU” Example

```
public class HelloHKU {
   public static void main(String[] args) {
      System.out.println("Hello HKU!");
      System.out.println("Welcome to COMP7506!");
   }
}
```

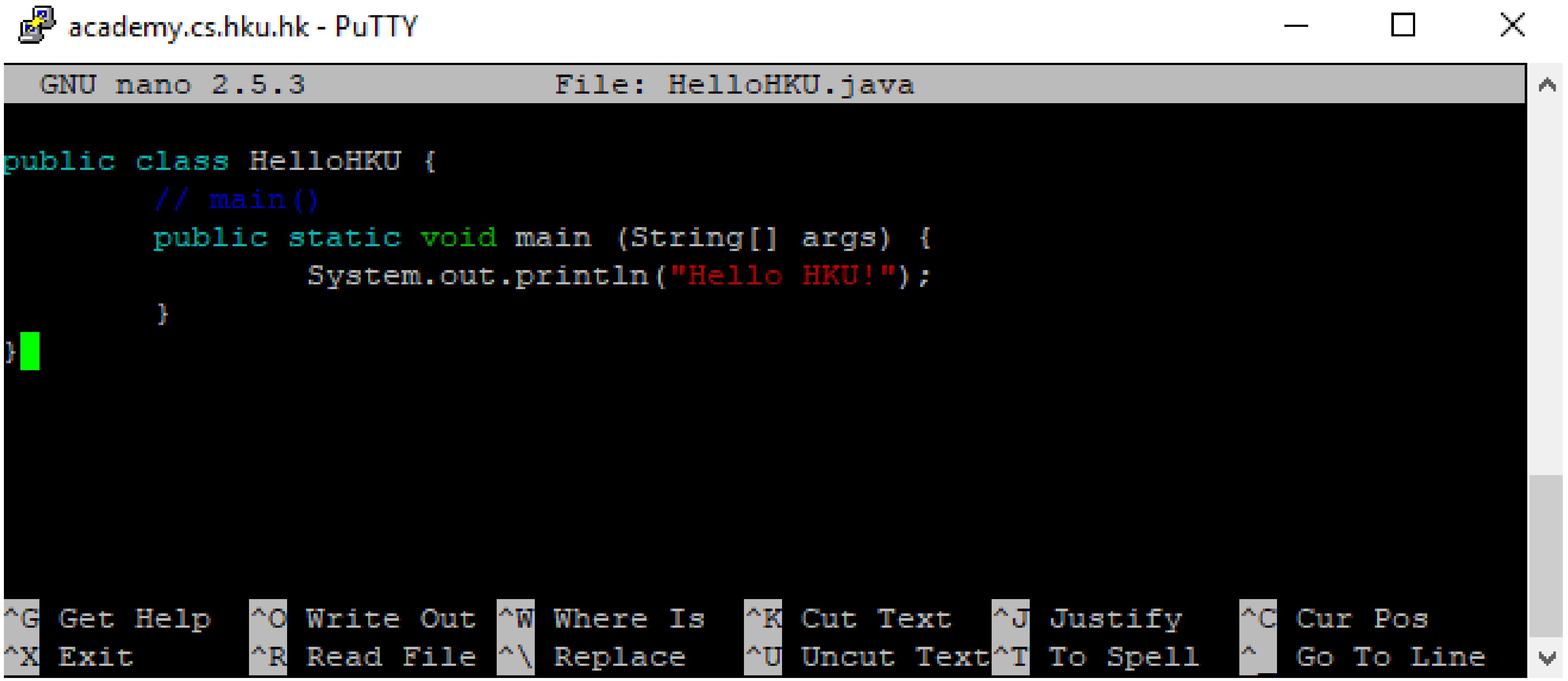
statements in the method

this statement simply prints its argument to the standard output

string to be printed

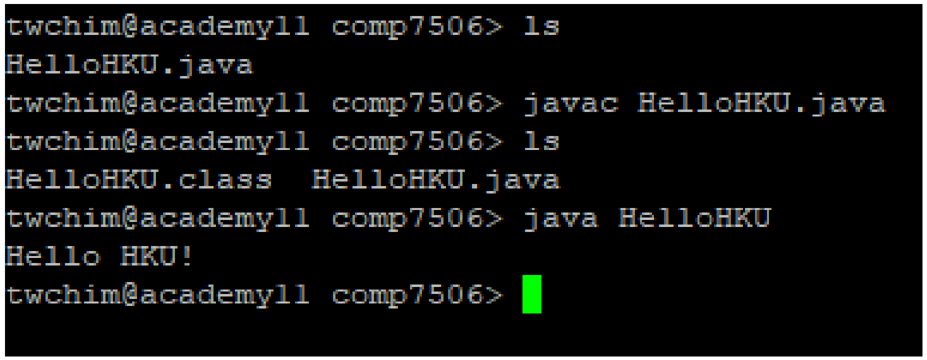
each statement ends with a semi-colon ;

"HelloHKU" Example



academy.cs.hku.hk - PuTTY

```
  GNU nano 2.5.3              File: HelloHKU.java

public class HelloHKU {
        // main()
        public static void main (String[] args) {
                System.out.println("Hello HKU!");
        }
}




^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text^T To Spell      ^  Go To Line
```



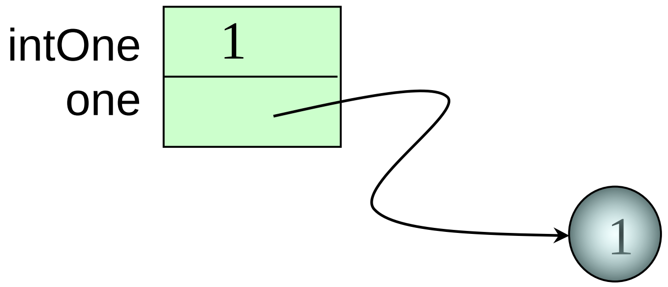
```
twchim@academy11 comp7506> ls
HelloHKU.java
twchim@academy11 comp7506> javac HelloHKU.java
twchim@academy11 comp7506> ls
HelloHKU.class  HelloHKU.java
twchim@academy11 comp7506> java HelloHKU
Hello HKU!
twchim@academy11 comp7506> 
```

Revision on Java Syntax

Primitive Types in Java

Primitive type	**Size**	**Minimum**	**Maximum**	**Wrapper type**
boolean	1-bit	0	1	**Boolean**
char	16-bit	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8-bit	-128	+127	**Byte**
short	16-bit	-2^{15}	$+2^{15}-1$	**Short**
int	32-bit	-2^{31}	$+2^{31}-1$	**Integer**
long	64-bit	-2^{63}	$+2^{63}-1$	**Long**
float	32-bit	IEEE754	IEEE754	**Float**
double	64-bit	IEEE754	IEEE754	**Double**

Wrapper Classes



- Java is an OOP language. Sometimes we want to treat a primitive value like an object, e.g., to keep a primitive value in an ArrayList of *Object*.
- For every primitive type, there is a wrapper class defined in *java.lang* whose object merely stores a value of the type. Many useful instance and static methods are provided in these wrapper classes.
- For *int*, the corresponding wrapper class is *Integer*.

```
int intOne = 1;
Integer one = new Integer(1);
System.out.println( one.equals(intOne) );
```

```
true
```

Standard Naming Conventions

- Professional programmers usually adopt standard naming conventions in their programs. These conventions also apply to mobile application development.

  - A class name begins with a capital letter (e.g., Dog)
  - A variable or method name begins with a lowercase letter (e.g., size, makeNoise())
  - The name of a symbolic constant consists of only capital letters (e.g., PI, GRAVITY)
  - Names consisting of multiple words are joined together with each subsequent word begins with a capital letter (e.g., ShoppingCart, addItemToCart())

Manipulation of Values

- Declare a variable: <Data Type> <Variable Name>

```
int var1;
char var2;
```

- Initialize a variable / assigning a value to a variable: <Variable Name> = <Initial Value>

```
var1 = 1;
var2 = ‘a’;
```

- Can declare and initialize at the same time: <Data Type> <Variable Name> = <Initial Value>

```
int var1 = 1;
char var2 = ‘a’;
```

Reference vs. Primitive

- Java handle objects and arrays always by reference (pointing to memory location).
  - Classes and arrays are known as reference types.
  - Class and array are composite type, don’t have standard size
- Java always handle values of the primitive types directly
  - Primitive types have standard size, can be stored in a fixed amount of memory.
- Because of how the primitive types and objects are handled, they behave differently in two areas: copy value and compare for equality.

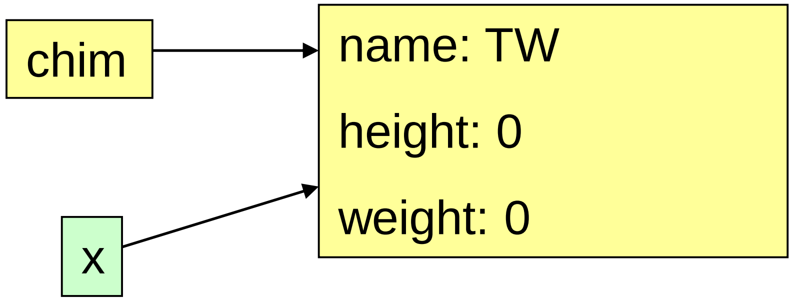
Copy

- Primitive types get copied directly by “=“

```
int x = 10; int y = x;
```

- For objects and arrays, we just copy the reference (still only one copy of object exists in heap memory).

```
Person chim = new Person();
chim.name = “TW”;
Person x = chim;
x.name = “Tim”;
System.out.println(chim.name);  // print “Tim”!
```



Loops

- 3 essential components: initial counter value, ending condition and advancing counter value.
- "for" loop:

```
for (int i = 0; i < 10; i++)
        System.out.println(i);
```

- "while" loop:

```
int i = 0;
while (i < 10) {
        System.out.println(i);
        i ++;
}
```

```
int i = 0;
do {
        System.out.println(i);
        i ++;
} while (i < 10);
```

Difference: If i = 10, left → do not print 10, right → print 10.

Array

- Define an array to store elements (same for primitive types and classes):

```
int[] myIntArray = new int[3];
int[] myIntArray = {1,2,3};
int[] myIntArray = new int[]{1,2,3};

String[] myStringArray = new String[3];
String[] myStringArray = {"a","b","c"};
String[] myStringArray = new String[]{"a","b","c"};
```

ArrayList

- ArrayList (need to import the library java.util.ArrayList)
- Define an ArrayList:

  ```
  ArrayList<Class> <Variable Name> = new ArrayList<Class>();
  ```
- Add elements into ArrayList:

  ```
  <Variable Name>.add();
  ```
- Get element at index i from ArrayList:

  ```
  <Variable Name>.get(i);
  ```
- Return index of object o or -1 (not found):

  ```
  <Variable Name>.indexOf(Object o);
  ```
- Get size of (number of elements in) ArrayList:

  ```
  <Variable Name>.size();
  ```

Array vs. ArrayList (Example)

Array:

```
public class Sample1 {
    public static void main(String[] args) {
        int[] A = new int[5];
        for (int i = 0; i < 5; i ++)
            A[i] = Integer.parseInt(args[i]);
        for (int i = 0; i < 5; i ++)
            System.out.println(A[i]);
    }
}
```

ArrayList:

```
import java.util.ArrayList;
public class Sample2 {
    public static void main(String[] args) {
        ArrayList<Integer> B = new ArrayList<Integer>();
        for (int i = 0; i < 5; i ++)
            B.add(Integer.parseInt(args[i]));
        for (int i = 0; i < B.size(); i ++)
            System.out.println(B.get(i));
    }
}
```

Features:

- No need to specify the size of array in advance
- Has the simple add() and get() interfaces
- Can refer to size using size() interface
- Very useful for classes

OOP Concept

Object-Oriented Programming

- Also known as OOP
- Terms relating to OOP
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Java programs contain nothing but definitions and instantiations of classes
  - Everything is encapsulated in a class!

Advantages of OOP

- Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear
- Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system
- Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods
- Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones
- Maintainability: objects can be maintained separately, making locating and fixing problems easier
- Re-usability: objects can be reused in different programs

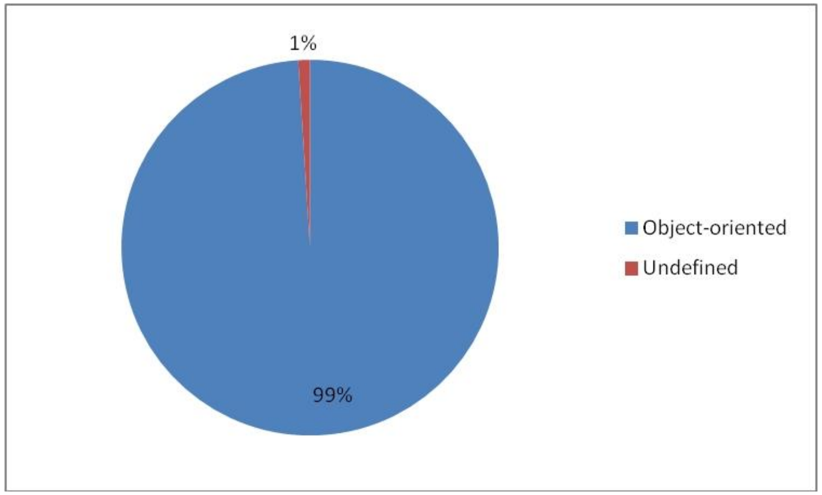
Industries with Highest Demand for OOP Developers

- Financial services
- Healthcare
- High tech
- Professional services
- Real estate
- Retail and e-commerce

- Reference:
  - “4 Advantages of Object-Oriented Programming” by Robert Half on Nov 18, 2021 (https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming)

Importance of OOP in Mobile Apps Development

- All major mobile device platforms (Android, iPhone, Windows and BlackBerry) uses Java / Kotlin, Objective-C / Swift, .NET, which are all object-oriented.
- These four platforms make up 99% of the smartphone market, and thus an overwhelming part of the mobile marketplace.
- The figure below shows how important object-oriented languages are to mobile development. Virtually, all of the mobile app development platforms are object-oriented.



Breakdown of Languages
Used in Mobile Development

Reference:
https://www.informit.com/articles/article.aspx?p=2036576

Java and Kotlin Examples

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

MainActivity.kt

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate
                (savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

- **MainActivity** is a class. An instance will be created when the app is run.
- **MainActivity** is a subclass of the pre-defined **AppCompatActivity** (where basic behaviors of all Android activities are defined).
- **onCreate()** is overridden so that we can define our own behavior.

Java and Kotlin Examples

MainActivity.java

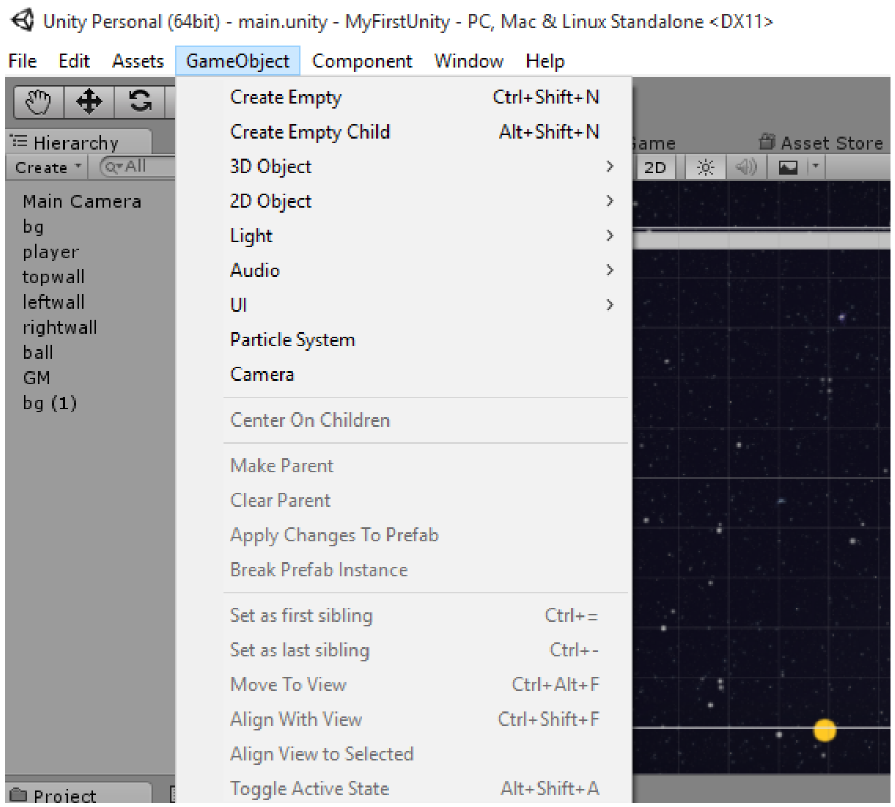
```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

MainActivity.kt

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate
                (savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

- **MainActivity** is a class. An instance will be created when the app is run.
- **MainActivity** is a subclass of the pre-defined **AppCompatActivity** (where basic behaviors of all Android activities are defined).
- **onCreate()** is overridden so that we can define our own behavior.

Unity Example



```
public class Cenemy {
   private float HP;
   void LostHP() {
      HP -= 10;
   }
   float GetHP() {
      return HP;
   }
}
public class CRedenemy : Cenemy {
   private Color color;
   public CRedenemy() {
      color = Color.red;
      …
   }
}
```

Objects can be found everywhere!

Principles of OOP

- Abstraction
  - The act of representing essential features without including background details
- Encapsulation
  - The act of wrapping up of data and operations into a single unit
- Inheritance
  - The process by which objects of one class acquire the properties of objects of another class
- Polymorphism
  - The ability to take more than one form

Inheritance

```
public class Employee {
        public string name;
        public int emp_no;
        public void Getname( );
        .........
}
        public class HP-employee extends Employee {
                private int hours;
                .........
        }

                public class MS-employee extends Employee {
                        .........
                }
```

Inherites name, emp_no and Getname() (HP-employee)

Inherites name, emp_no and Getname() (MS-employee)

Polymorphism

“Standard” object declaration and assignment:

```
Dog myDog = new Dog();
```

1. Dog myDog: declare a reference variable
2. new Dog(): create a new object
3. =: point the reference to the object

The reference type and the object type are the same.

“Non-standard” object declaration and assignment (polymorphism):

```
// Animal is superclass of Dog
Animal myDog = new Dog();
```

The reference type can be a superclass of the actual object type.

Polymorphism

```
// Animal is superclass of Dog, Cat, Wolf, Hippo and Lion
Animal[] animals = new Animal[5];
animals[0] = new  Dog();
animals[1] = new  Cat();
animals[2] = new  Wolf();
animals[3] = new  Hippo();
animals[4] = new  Lion();

for (int i = 0; i < animals.length; i++)
{
    animals[i].eat();
    animals[i].roam();
}
```

When i is 0, animals[0] is a Dog, Dog's eat() and roam() are called

When i is 1, animals[1] is a Cat, Cat's eat() and roam() are called

Concept of “Class”

- Fundamental unit of OOP program
- A class is a template or blueprint for objects
- A class describes a set of objects that have identical characteristics and behaviors (methods).
  - E.g., 1: Existing classes provided by the language (e.g., AppCombatActivity)
  - E.g., 2: User defined classes (e.g., Student)
- Each class defines a set of fields (variables), methods or inner classes.

Concept of “Object”

- An object is an instance of a class.
- An object has behavior as defined in the class (methods).
- An object has its own state (instance variables).
- An object is a chunk of memory in the heap (e.g., holding field values) and has unique address.
- For example, **Robot** is a class. You can define **R1**, **R2**, …, **RN** as instances of the class **Robot** and they are stored at different heap memory locations.

Recall:

- **Heap memory: dynamically created objects**
- **Stack memory: method invocations, local variables, etc.**

Scoping in Class

```
public class VisibilityDemo {
    private int classVar1;
    private int classVar2;
    public int method1 (int x) {
        int local = 0;
        for (int i = 0; i < x; i++) {
            local += i;
        }
        return local;
    }

    public void method2 (int x) {
        classVar1 = x + 10;
        classVar2 = method1(classVar2);
    }
}
```

The red identifiers denote class variables and methods. They have visibility anywhere inside the outermost pair of red curly brackets.

The blue identifiers are local to a single block (identified by blue brackets). They are not visible to anything outside of their block, but are visible inside blocks nested inside of the blue bracketed block.

The gray identifiers are found inside the for-loop. The gray variable *i* is visible only inside the loop.

Parameters are denoted by green. They are visible everywhere inside the method in which they appear, but only in that method.

The “static” keyword

- Java methods and variables can be declared static
- Static variables are **independent of any object**
- This means that a Class’s
  - static methods can be called even if no objects of that class have been created and
  - static data is “shared” by all instances (i.e., one value per class instead of one per instance

```
class StaticTest {static int i = 47;}
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
// st1.i == st2.i == 47
StaticTest.i ++;    // or st1.i ++; and st2.i ++;
// st1.i == st2.i == 48
```

Chapter 4.

End

2023-2024
COMP7506 Smart Phone Apps Development
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong