LESSON V.
Object Initialization and Usage

Vu Thi Huong Giang

SoICT (School of Information and Communication Technology)

HUST (Hanoi University of Science and Technology)

Objectives

- Acquaint how to initialize and use objects

Content

- Data initialization and constructor
- Object declaration and initialization
- Object usage

I. Data initialization

- Needs of initialize data before using
- Primitive data type: initialize by assignment operator
- Object (reference data type): initialize by using constructor.

Constructor

- Objects are created through constructors
  - instance variables of an object are initialized in the scope of constructors.
- A class may declare its own constructor or rely on the default constructor provided by the Java environment.
  - The name of constructor is the name of class
  - Constructor is written without return type; the default return type of a class constructor is the same class

Constructor's definition: without parameter

- Fixing the initial values of attributes
- Syntax:

```
modifier class-name() {
    // constructor body
}
```

- Default constructor:
  - constructor without argument.
  - It is automatically provided in the case of no explicit declaration
  - Initialize attributes with the default values of the corresponding data types

- Example

```
public class Account {
     private String owner;
     private long balance;
     public Account() {
          owner = "Noname";
          balance = 100;
     }
}
```

- If this constructor is not implemented, the java default constructor will initiate attributes with the following values:
  - owner = null (Default value of String type)
  - balance = 0 (Default value of long type)

Constructor's definition: with parameter

- Parameterizing the initial values of attributes

```
modifier class-name(parameter-list) {
  // constructor body
}
```

- Example

```
public class Account {
    private String owner;
    private long balance;
    public Account(String name, long money) {
        owner = name;
        balance = money;
    }
}
```

II. Object declaration and initialization

```
datatype instance-variable;
instance-variable = new datatype();
```

or

```
datatype instance-variable = new datatype();
```

- Declaration: declare a reference variable
  - Associate a variable name with a **`datatype`** object
- Instantiation: **`new`** is a Java operator that creates the object (i.e. creates an instance of class **`datatype`**)
  - Allocate memory for a **`datatype`** object
  - Return its address
- Initialization: initialize the new object
  - Call to a constructor

Object initialization

- Syntax

  `datatype instance-variable = new datatype(…);`
- When the object is created, the member variable is assigned to the memory area, and initialized at the same time.
- Implicit initialization:
  - number data type ← 0;
  - reference type ← null
  - boolean ← false

  (see the default values of data types, lecture 2)

III. Object usage

- Using an object implies
  - Accessing (taking or changing) the value of one of its variables
  - Calling one of its methods to perform an action.
- Objects communicates through message passing

  `receiver.message`
  - The dot operator (“.”) is used to send a **`message`** to **`receiver`** object

1. Accessing and changing the value of a variable

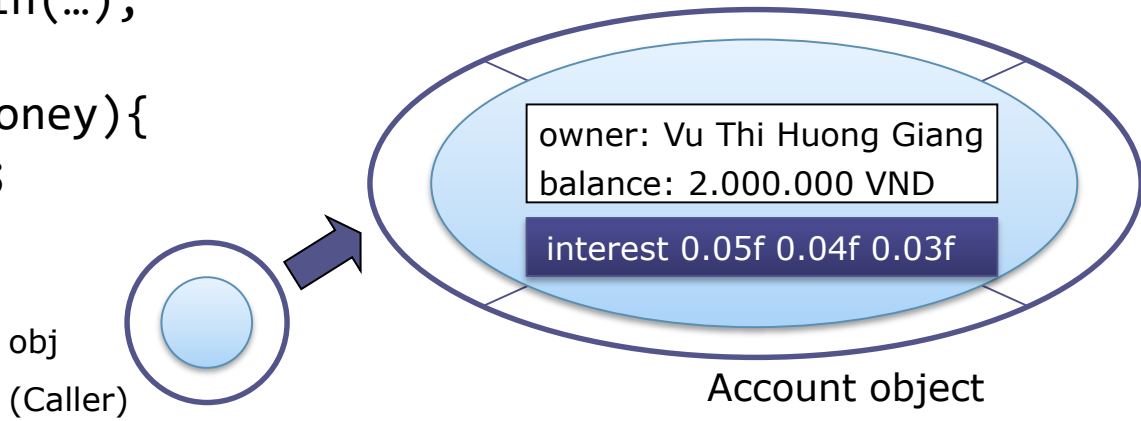
receiver.member-variable-name;

- The message is the variable name
- No parameter is used
- No need to use the dot operator inside a class

```
public class Account {
    String name; //Account name
    long balance; //Balance

    void display(){
        System.out.println(…);
    }
    void deposit (long money){
        balance += money;
    }
}
```

```
// Class that uses
// the variable balance
Account obj = new Account();
obj.balance = 100;
```



2. Calling methods

- A method is called through an object
  - The object is its default target

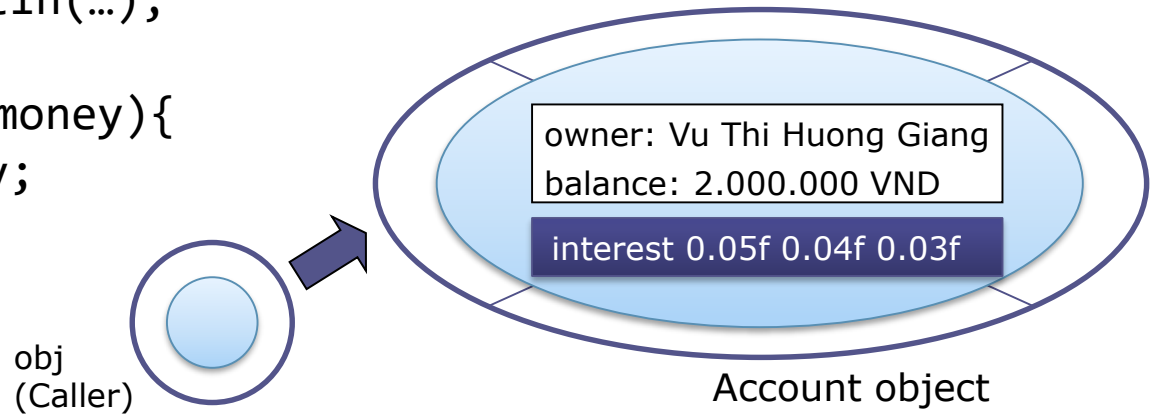
receiver.method-name(list-of-parameters)

  - The message is the method name.
  - The parameters respect the signature of method.

```
public class Account {
    String name; //Account name
    long balance; //Balance

    void display(){
        System.out.println(…);
    }
    void deposit (long money){
        balance += money;
    }
}
```

```
// Class that uses
// methods of Account object
Account obj = new Account();
obj.display();
obj.deposit(1000);
```



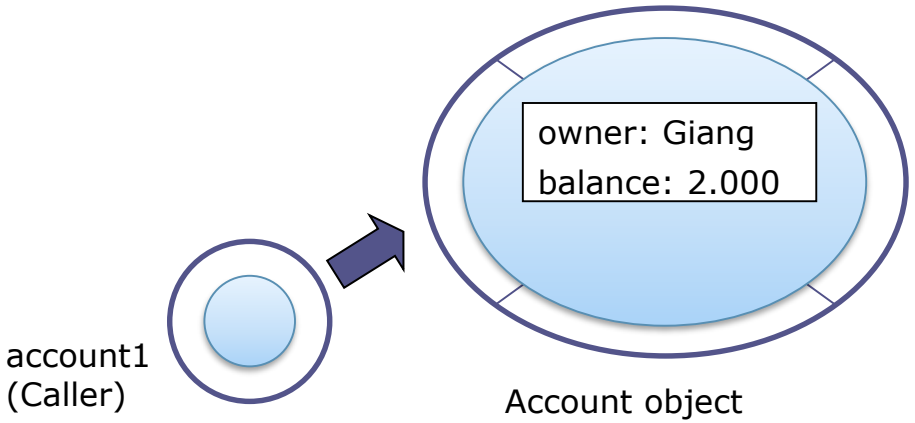
3. Calling constructors

- A constructor can not be called directly
- It can only be called by using the new operator during class instantiation.

```
public class Account {
     // Account name
     private String owner;
     // Account name
     private long balance;

     public Account
        (String name, long money ) {
        owner = name;
        balance = money;
     }
}
```

```
public class AccountUsage {
    public static void main(String[] args) {
        // Object creation
        Account account1 =
            new Account("Giang", 2000);
    }
}
```



Example

```
public class Track {
    // attributes
    private String title;
    // title of the track
    private int time_length;
    // length of time for playback
    private int data_format;
    // encoding format
    private int buffer_size;
    // size of the buffer where data is
    // read from for playback

    // constructor without parameter
    public Track() {
          title = "Notitle";
    }
```

```
    public Track(int length, int format, int size) {
          title = "Notitle";
          time_length = length;
          data_format = format;
          buffer_size = size;
    }
    public Track(String name, int length, int format,
    int size) {
          title = name;
          time_length = length;
          data_format = format;
          buffer_size = size;
    }
    public void setLength(int length) {
          time_length = length;
    }
    public void setTitle(String name) {
          title = name;
    }

    // ....

}
```

Example

```
public class TrackClassUsage {
     public static void main(String[] args)
        {
          Track track1 = new Track();
          track1.setTitle(“One name");
          track1.setLength(45);
          Track track2 = new Track(
            "Four seasons", 43, 1, 1000);
          Track track3 = new Track(
            “One name", 45, 0, 0);
          Track track4 = track3;
          track2 = new Track(
            “Another name", 0, 0, 1000);
     }
}
```

- Objects are used usually through variables
- Track1 and track3 have the same initial values, but they are not the same object
- Track3 and track4 refer to the same object
- The object ("Four seasons", 43, 1, 1000) is no longer accessible

4. Keyword `this`

- Used inside a method or a constructor to refer to the current object
  - Specify member or method of current object
  - Distinguish the data member with the parameters of member functions (which have the same name)

```
public class Account {
     // instance variable
     private String owner; // Account name
     private long balance; // Balance
     //...
     // value setting method
     public void setAccountInfo(String owner, long balance) {
          this.owner = owner;
          this.balance = balance;
     }
     //...
}
```

Keyword this

- Used to call another constructor of own class

```
public class Track {
      private String title; // title of the track
      private int time_length; // the length of time to playback
      private int data_format;
      // the encoding format in which the data (audio, video, text...) is represented
      private int buffer_size;
      // the total size of the buffer where data is read from for playback

      public Track() {
            this(0,0,0);
      }

      public Track(int length, int format, int size) {
            title = "Notitle";
            time_length = length;
            data_format = format;
            buffer_size = size;
      }
}
```

'this'(argument list) calls another constructor

Keyword **this**

- Used to pass the current object's reference to other objects

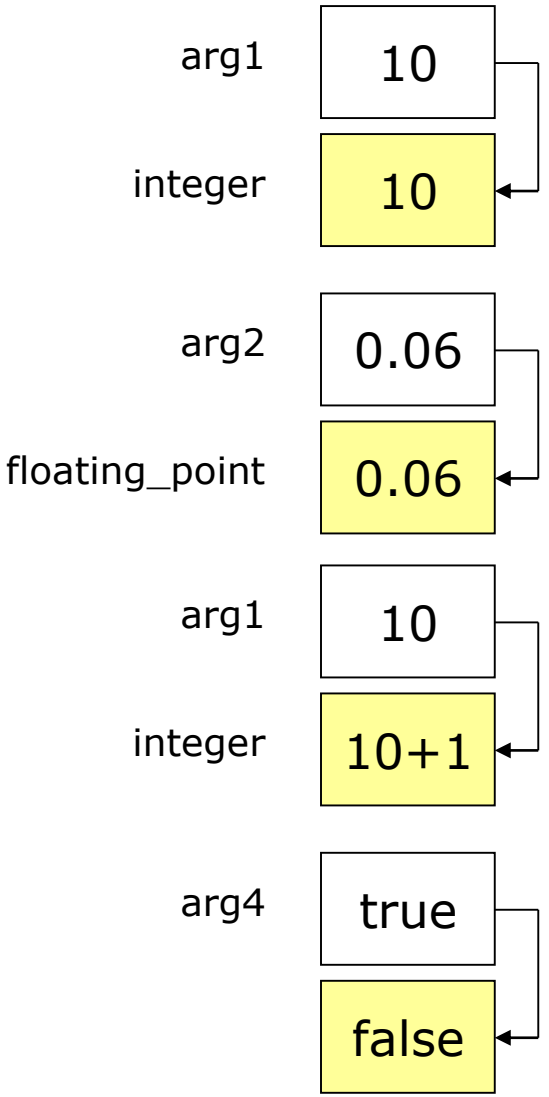
```
public class Track {
     private String title; // title of the track
     private int time_length; // the length of time to playback
     private int data_format;
     // the encoding format in which the data (audio, video, text...) is represented
     private int buffer_size;
     // the total size of the buffer where data is read from for playback
     public Track() {
          this(0,0,0);
     }
     public Track(int length, int format, int size) {
          title = "Notitle";
          time_length = length;
          data_format = format;
          buffer_size = size;
          RecordException re = new RecordException(this);
     }
}
```

5. Argument-passing

- Parameter: a variable receiving value at the time the method is called
- Argument: a value passed to the method when it is called
- Two ways of how the arguments are passed to methods:
  - Parameters of primitive type: by value
    - a method receives a copie of the original value;
  - Parameters of reference type: by reference
    - a method receives the memory address of the original value, not the value itself

Passing arguments by value

```
public class PassingByValueUsage {
     static int integer;
     static float floating_point;
     char character;
     boolean logic;
     public static void
        TestPrimitiveParam(int arg1,
        float arg2, char arg3,
        boolean arg4) {
          //passing by value
          //new values can be seen
          //outside the enclosed method
          integer = arg1;
          floating_point = arg2;
          // legal assignment
          // new values can not be seen
          // outside the enclosed method
          arg1 += arg1;
          arg4 = false;
     }
```



Passing arguments by value

```
public class PassingByValueUsage {
    static int integer;
    static float floating_point;
    char character;
    boolean logic;
    public static void
       TestPrimitiveParam(int arg1,
       float arg2, char arg3,
       boolean arg4) {
         //passing by value
         //new values can be seen
         //outside the enclosed method
         integer = arg1;
         floating_point = arg2;
         // legal assignment
         // new values can not be seen
         // outside the enclosed method
         arg3 += arg3;
         arg4 = false;
    }
    public static void main(String[] args) {
         int arg1= 10;
         float arg2 = 0.06f;
         char arg3 = 'a';
         boolean arg4 = true;
         System.out.println(
           PassingByValueUsage.integer + " " +
           PassingByValueUsage.floating_point);
         System.out.println(arg1 + " " + arg2 +
           " " + arg3 + " " + arg4);

         TestPrimitiveParam(
           arg1, arg2, arg3, arg4);

         System.out.println(
           PassingByValueUsage.integer + " " +
           PassingByValueUsage.floating_point);
         System.out.println(arg1 + " " +  arg2
           + " " + arg3 + " " + arg4);
    }
}
```

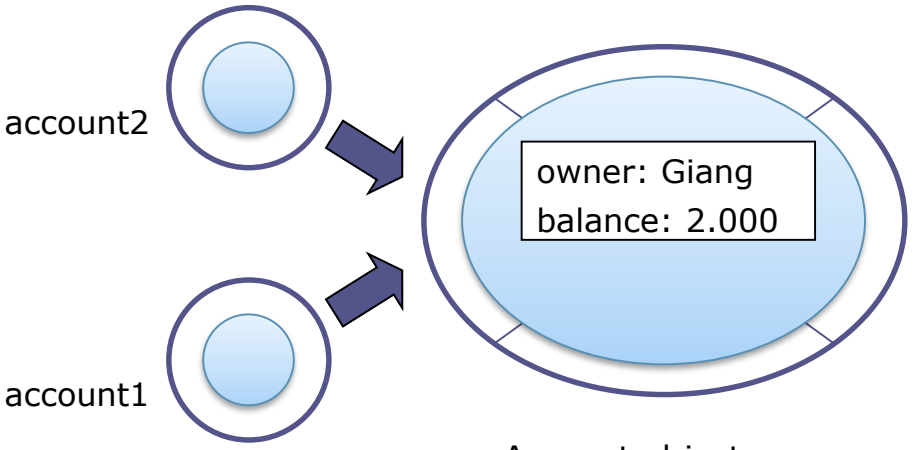
Passing arguments by object reference

```
public class Account {
     // Account name
     private String owner;
     // Account name
     private long balance;

     public Account
        (String name, long money ) {
        owner = name;
        balance = money;
     }
}
```

```
public class AccountUsage {
    public long checkAccountBalance
        (Account acc){
        return tmp = acc.getBalance();
    }

    public static void main(String[] args) {
        // Object creation
        Account account1 =
            new Account("Giang", 2000);
        Account account2 = account1;
        long blc =
            checkAccountBalance(account2);
    }
}
```



account1 and account2
refer to the same object

Quiz

1. How to get the changed value inside a method ?
    1. Return the value that was changed.
2. Consider the Track class (slide 15). Modify and complete operations of setting/getting all of its attributes with the this keyword.
3. Implement a program for testing these operations and showing the results.
4. Consider the Track class. Implement different overloaded constructors (with and without parameters) for creating the Track objects in different ways.
5. Implement a program for testing these operations and showing the results.

Review

- Data must be initialized before used
- Constructor is used to create objects
- Object must be declared and initialized before used
- Object usage :
  - All objects are allocated and accessed through reference variables
  - this: refers to the current object
  - argument passing: by value and by reference