

# Introduction to Java

Objects and Classes

COMP2603

Object Oriented Programming 1

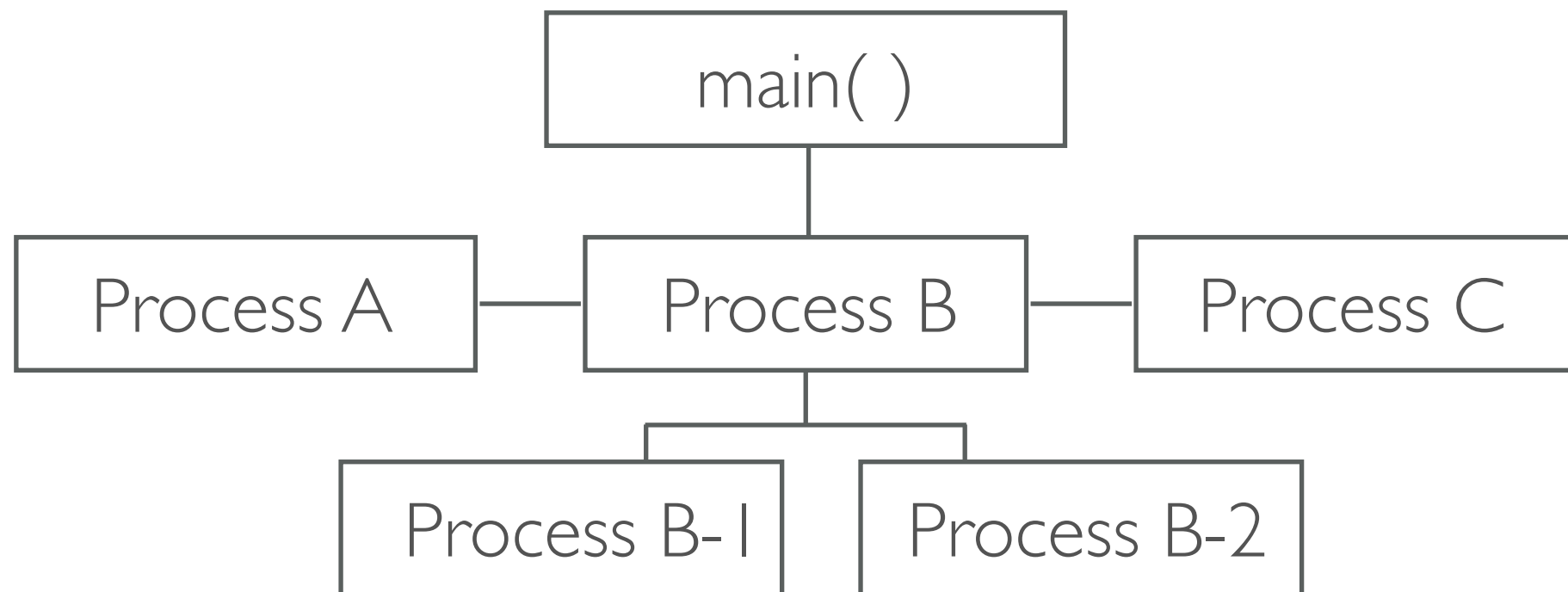
Week 1

# Outline

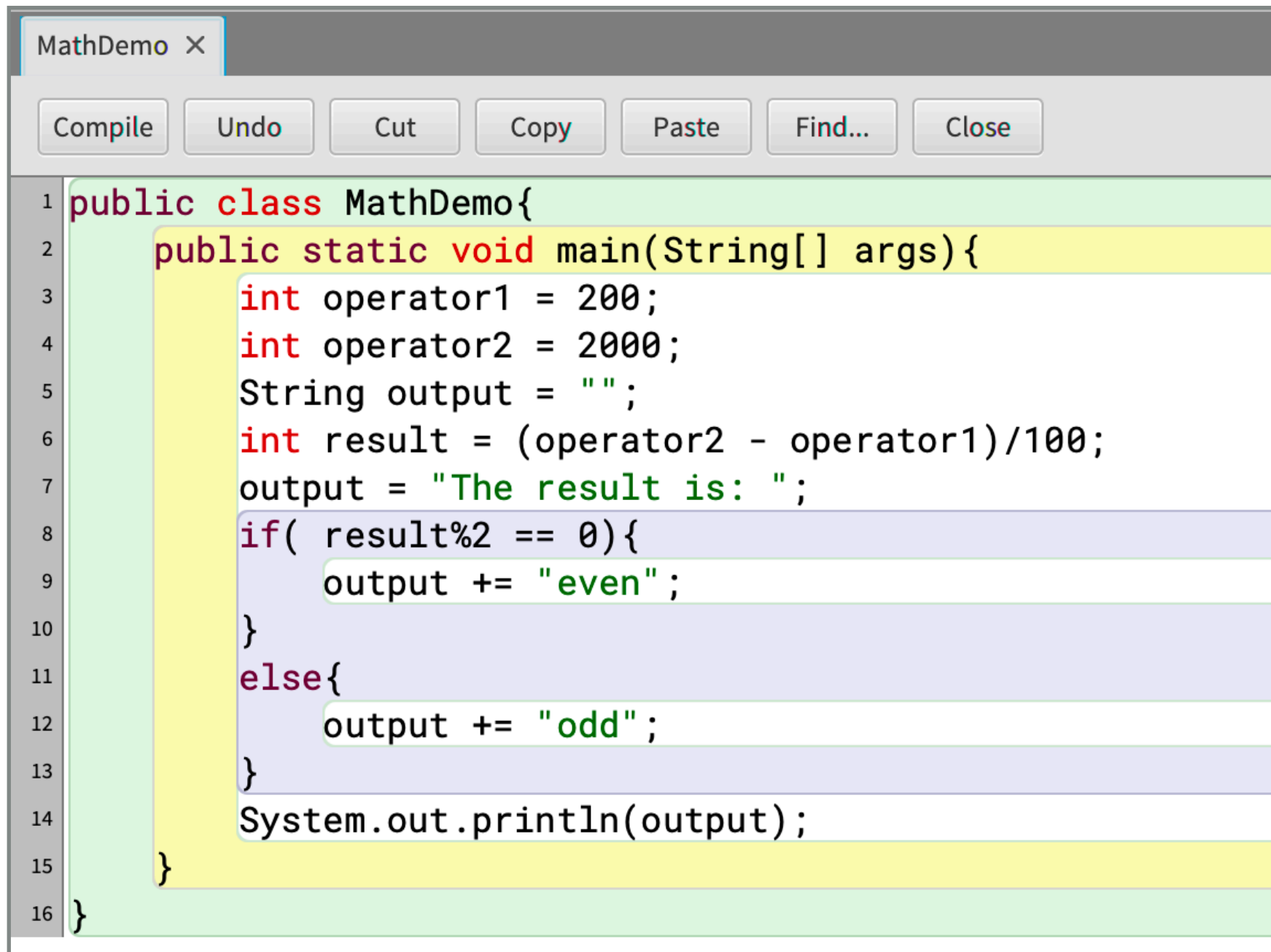
- Programming Paradigms
- Differences with C, C++
- Java Language Features
  - Primitive Data Types: Numeric, Character, Boolean
  - Arithmetic, Relational Operators
  - Class Types: Arrays, Strings
  - Conditional Statements
  - Loops, Switch Constructs

# Programming Paradigms

- **Procedural:** Consists of a set of processes that are hierarchically connected.



# A Procedural Java Program



```
1 public class MathDemo{
2     public static void main(String[] args){
3         int operator1 = 200;
4         int operator2 = 2000;
5         String output = "";
6         int result = (operator2 - operator1)/100;
7         output = "The result is: ";
8         if( result%2 == 0){
9             output += "even";
10        }
11        else{
12            output += "odd";
13        }
14        System.out.println(output);
15    }
16 }
```

Output:

The quotient is odd

# Programming Paradigms

## **Object-Oriented:**

Set of objects collaborating to achieve the goals of the application.

- Focuses on interactions between objects
- Iterative, incremental development
- Represents real-world problems and domains

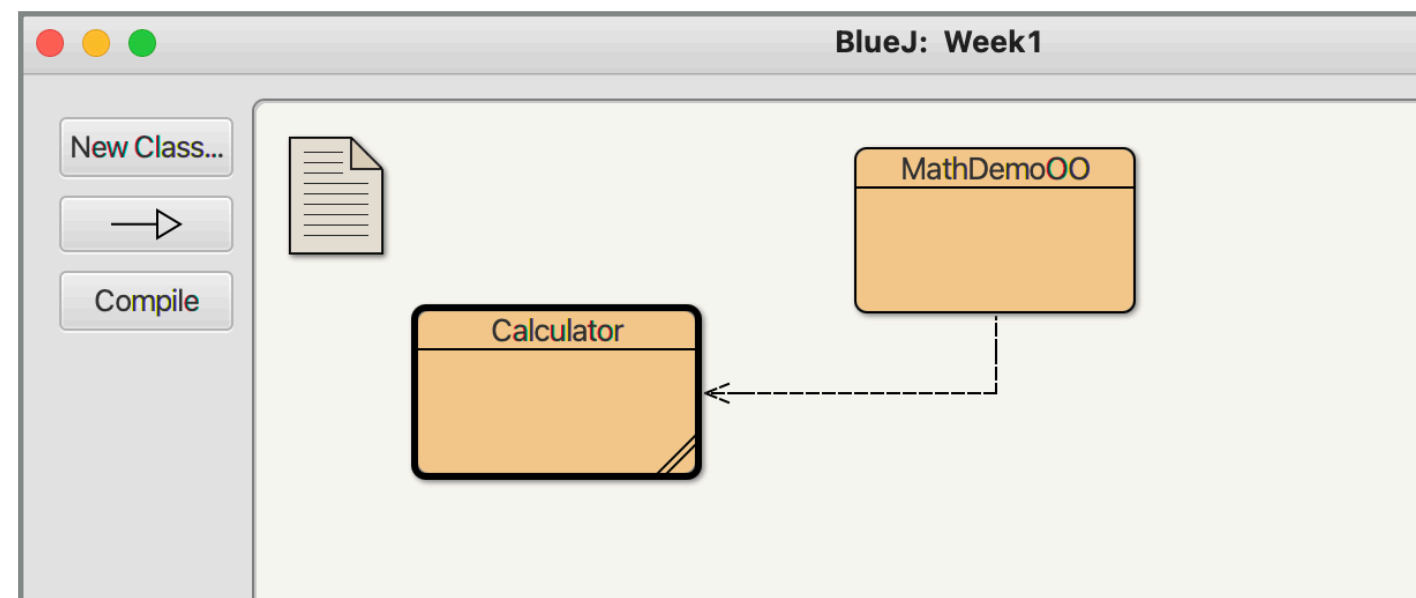
# An Object-Oriented Java Program

```
MathDemoOO X
[Compile] [Undo] [Cut] [Copy] [Paste] [Find...] [Close]

1 public class MathDemoOO{
2     public static void main(String[] args){
3         Calculator c = new Calculator();
4         int difference = c.subtract(200,2000);
5         int quotient = c.divide(difference,100);
6         if(c.isOdd(quotient))
7             System.out.println("The quotient is odd");
8         else
9             System.out.println("The quotient is even");
10    }
11 }
12 }
```

```
Calculator X
[Compile] [Undo] [Cut] [Copy] [Paste] [Find...] [Close]

1 public class Calculator{
2     public int subtract(int subtrahend, int minuend){
3         return minuend - subtrahend;
4     }
5     public int divide(int dividend, int divisor){
6         return dividend/divisor;
7     }
8     public boolean isEven(int number){
9         return (number%2 == 0);
10    }
11    public boolean isOdd(int number){
12        return (number%2 != 0);
13    }
14 }
```



Output:

The quotient is odd

# Object

- **Object:** a primary modelling element in OOP
- Distinct Entity
  - current state
  - well-defined behaviour
- Created from *Classes*

# Classes

- **Classes** define
  - **Attributes** : set of properties
  - **Behaviour**: common to all members of the class
- Template for creating objects of the same type
- Individual objects have different values for attributes
- Examples of Classes: Account, Student



# Attributes

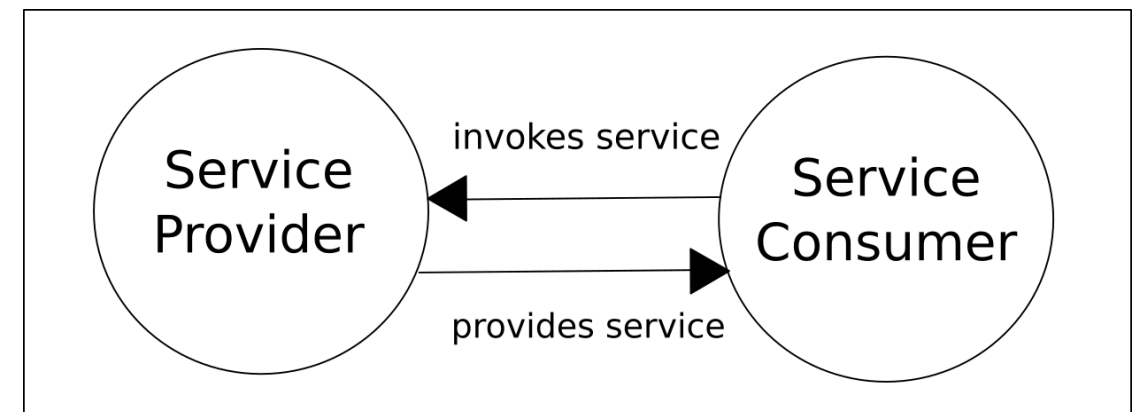
- Attributes are **properties** of an object
- Store data about an object
- The **state** of an object is the set of values of each of its attributes
- Typically given noun-like names e.g. number, balance, firstName, lastName
  - Match the purpose served in the app.
- Have types e.g. int, double, String

```
/* e.g. Account object
attributes */

int number;
double balance;
```

# Behaviour

- Regarded as a **service** provided by an object
- Particular action or task that the object performs
- Depends on the current state, sometimes it results in the modification of that state
- Implemented as **methods**
- Typically given verb-like names e.g. deposit, withdraw
  - Represents a particular task



# Implementing Behaviours

- **Accessors:**  
allow access to  
the state of an  
object

```
/* Account behaviour */

//Accessor for balance
public double getBalance(){
    return balance;
}

//Accessor for account number
public double getNumber(){
    return number;
}

//To String
public String toString(){
    String s;
    s = "Number:" + number + " Balance: " + balance;
    return s;
}
```

# Implementing Behaviours

- **Mutators:** modify the state of an object

```
/* Account behaviour */

//Deposit funds into account
public void deposit(double amount){
    balance = balance + amount;
}

//Remove funds from account
public void withdraw(double amount){
    if(balance >= amount)
        balance = balance - amount;
}
```

# Creating a Class

- A **class** is 'template' from which objects are created
- Particular combination of attribute values differentiate objects
- An **object** is referred to as an **instance** of a class

```
public class Account{

    //Declare attributes
    int number;
    int balance;

    //Declare and define methods
    public double getBalance(){
        return balance;
    }

    public double getNumber(){
        return number;
    }

    public String toString(){
        String s;
        s = "Number:" + number + " Balance: " + balance;
        return s;
    }

    public void deposit(double amount){
        balance = balance + amount;
    }

    public void withdraw(double amount){
        if(balance >= amount)
            balance = balance - amount;
    }
}
```

# Creating a Class

- **Access Modifier:** keywords that restrict access to the features of an object
  - **public**
  - **private**
- Classes are public so that other objects can create and use objects of the class
- Methods are public so that services can be invoked by another class
- Attributes are private because the object should control access of and updates to its state

```
public class Account{

    //Declare attributes
    private int number;
    private int balance;

    //Declare and define methods
    public double getBalance(){
        return balance;
    }

    public double getNumber(){
        return number;
    }

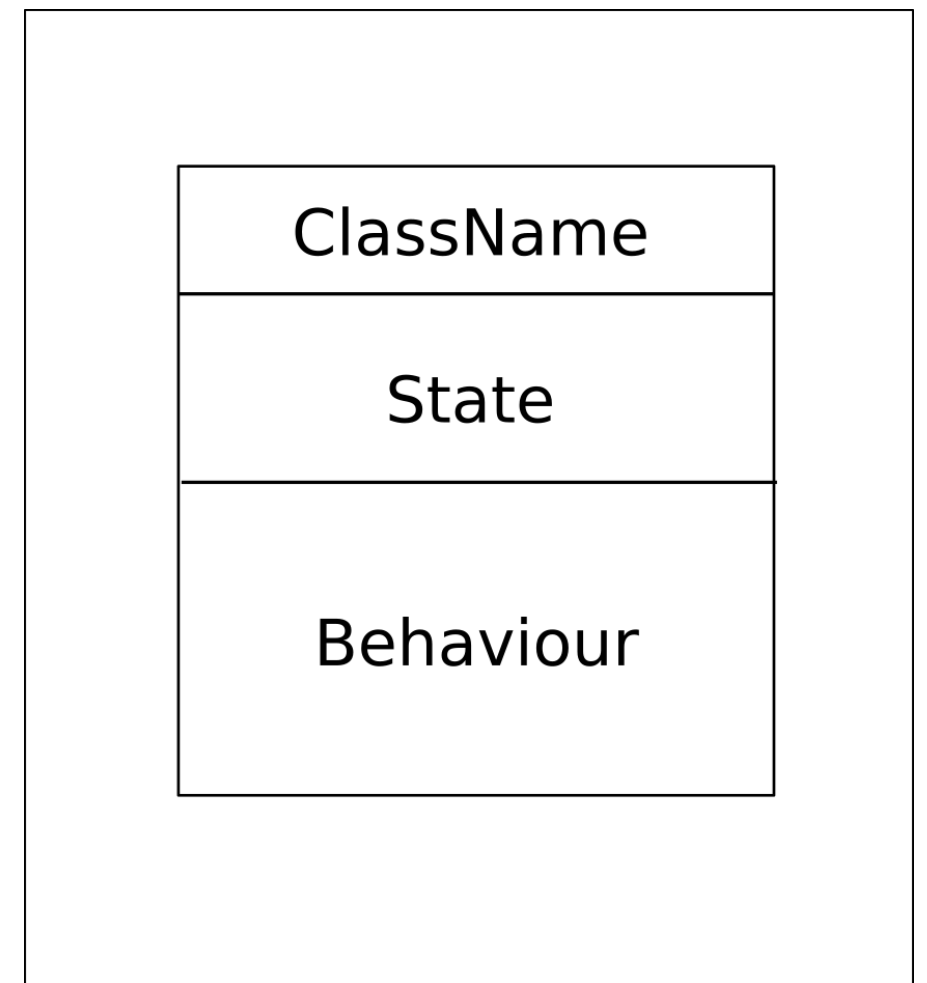
    public String toString(){
        String s;
        s = "Number:" + number + " Balance: " + balance;
        return s;
    }

    public void deposit(double amount){
        balance = balance + amount;
    }

    public void withdraw(double amount){
        if(balance >= amount)
            balance = balance - amount;
    }
}
```

# UML Notation

- UML: Unified Modelling Language
- Graphical Language
  - Widely used for OOP apps.
- Provides a visual representation of a class and the collaboration between objects of different classes



# UML Notation

- Class diagram: 3 components
  - Top: class name
  - Middle: attributes with type specifiers if necessary
  - Bottom: methods of the class

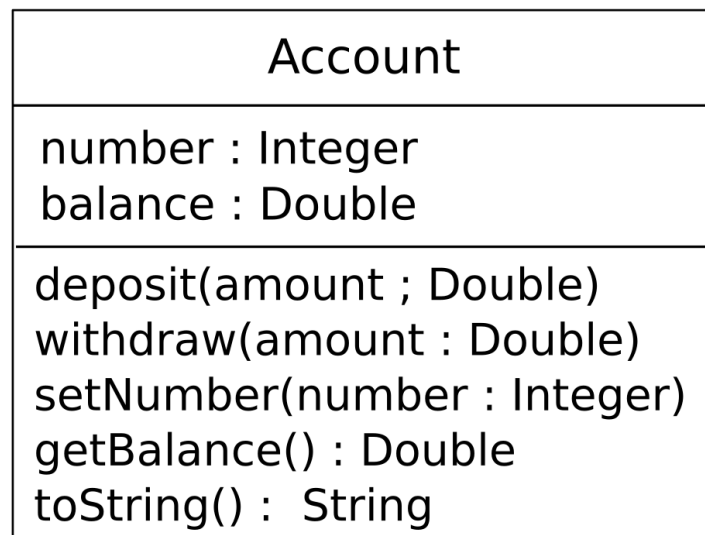
Account
number : Integer balance : Double
deposit(amount ; Double) withdraw(amount : Double) setNumber(number : Integer) getBalance() : Double toString() : String



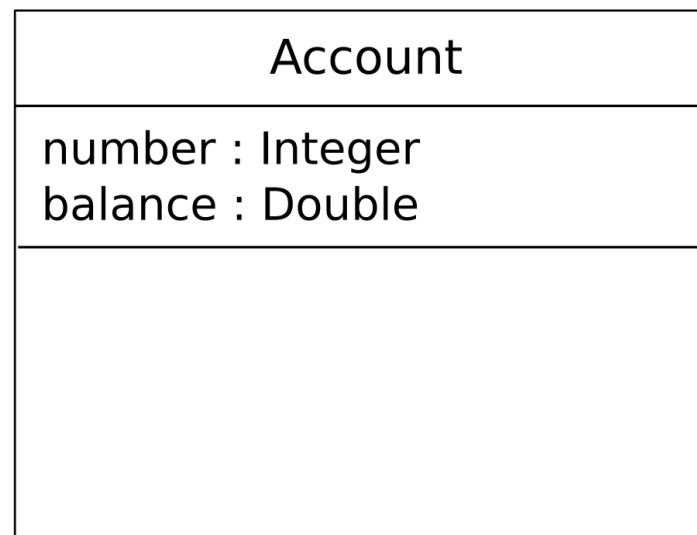
# UML Notation

## Class Diagram Variations

(a)



(b)



(c)



# Creating Instances

- An instance of a class must first be created in order to use the services of an object
- **Instantiating** the class : **new** keyword

```
new Account(); // Creates a new Account object
```

- Creates a new Account object in memory
- Need to refer to the object using an object variable

# Creating Instances

```
Account a;           // Declare an object variable  
a = new Account(); // Assign a new object to a
```

- Object variable *a* must be declared to be the same type as the object to which it will refer
- Reference is for one Account object

```
Account b;  
b = null;           // b does not refer to a specific  
                    // Account object right now.
```

# Manipulating Instances

```
a.setNumber(10); // Account has a number  
a.deposit (1000.00); // Deposit $1000.00 to account
```

- The object reference can be used to request services using method invocations
- Services request:  
*objectVariableName.methodName(arguments);*
- Client object supplies arguments

# Manipulating Instances

```
b.setNumber(20); // b is null
```

- *b* does not refer any object
- Requesting a service from *b* results in a serious programming error
- Compiler sometimes detects this, but if it occurs at run-time: `NullPointerException` and program halts

# Creating a Client Class

```
public class BankApplication{  
  
    public static void main(String[] args){  
        Account a;  
        a = new Account();  
        a.setNumber (10);  
        a.deposit(1000.00);  
        System.out.println(a.toString());  
    }  
}
```

- BankApplication.class file must be in the same folder as the Account.class file
- Complete program: Two collaborating classes.