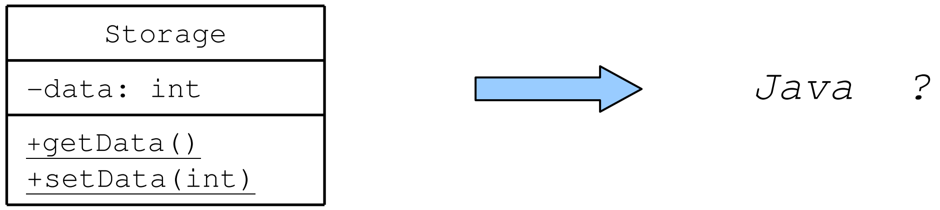
Java Classes and Objects

What you already know

- How to use the UML class symbol to describe classes including data **fields** and **methods**
- How to use UML class diagrams to describe **relationship between classes**
- How to declare and use **local variables** in methods to process data
- How **static methods** of existing classes can be called (*Example*: `Math` class methods)
- How to declare **reference variables** and **objects** of existing classes (*Example*: `Scanner` methods)
- How to input values from the **command line** and the **keyboard input buffer**
- How to display values in various formats

What you need to know

- How to define a new class in a Java program based upon its UML description



- How to define methods in a Java class according using its UML description
- How methods can implement class behaviours
- How to initialise data fields when a new object is created
- How to implement relationship between classes in Java programs

Limitations of `main()` method

- The main() method is where program execution starts
- According to the Java language specification the *main()* method must have the following signature

```
public static void main( String[] args )
{
    .   .   .
}
```

As the method is declared as static:

- you cannot use it to access non static data fields defined in the class
- you cannot call non-static methods declared in the class

- The only option to implement the program functionality is to put all code in *main(),* or other static methods defined in the same class…

What can be done to overcome this problem?

Defining another method

- To understand how objects in a Java application can interact, you need to understand first how methods can interact
- To start with, lets' consider a simple example with two static methods

```
class Example
{
   public static void main(String[] args)
   {
       System.out.println("This is main() method");
       sayHello();
       System.out.println( "This is main() method again" );
   }

   public static void sayHello()
   {
       System.out.println("  > This is sayHello() method - Hello!");
       return;
   }
}
```

1. Execution of main() is suspended
2. Control is passed to sayHello()

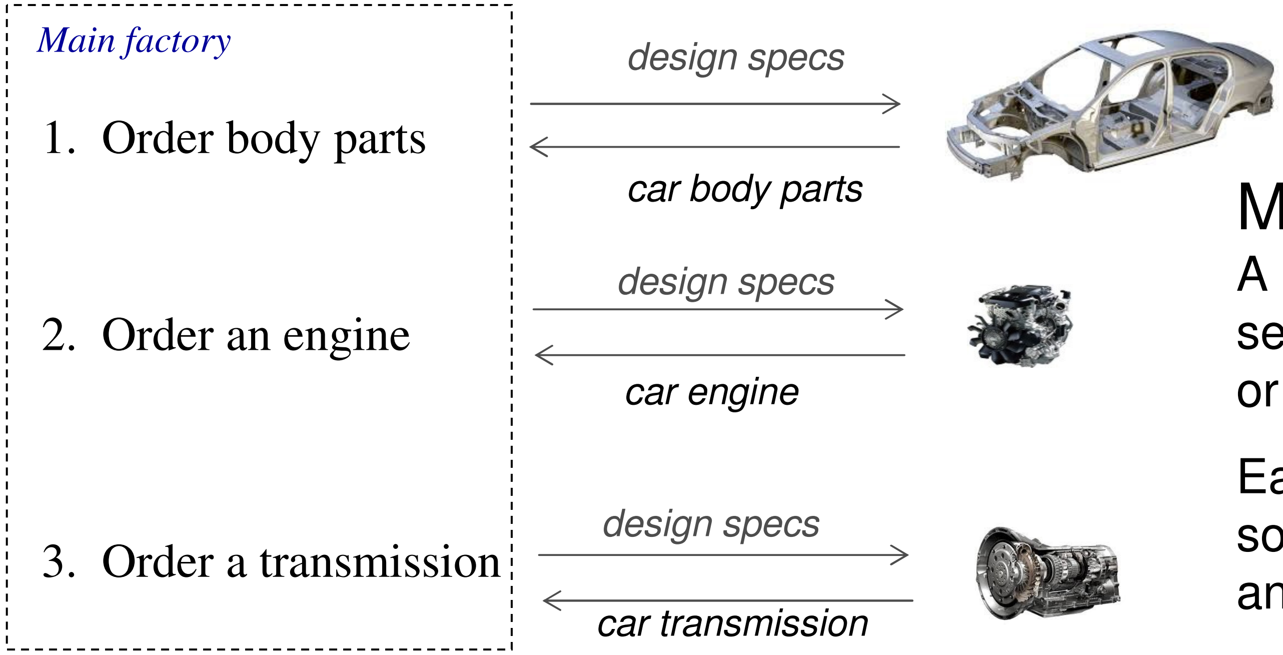
Control is returned back to main()

```
This is main() method
  > This is sayHelo() method - Hello!
This is main() method again
```

Why should I define methods?

- Unlike simple examples discussed earlier, classes defined in real projects implement complex behaviours
- It is a very poor practice to put all code in one method (for example in **`main`**). Different behaviours should be implemented in different methods (Modular design)

Example: Make a car. Making everything at one factory is not efficient



Modular design:

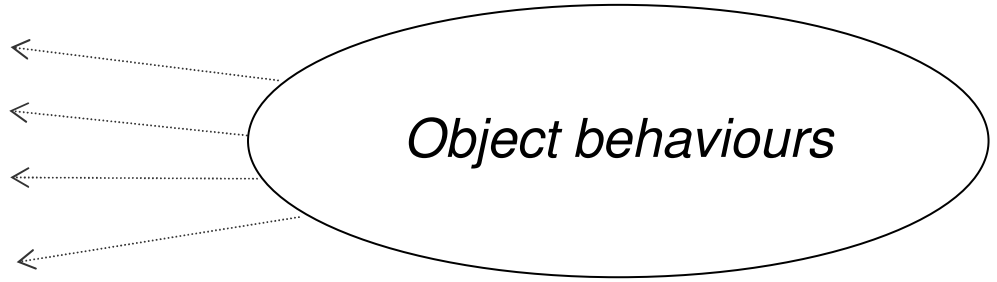
A big job is subdivided into several well defined **tasks** or **procedures**

Each procedure requires some **input** information and **returns** the result

What should methods do?

- When you define a method you need to decide what object behaviour it will be responsible for
  - support user input
  - carry out calculations
  - set private data fields
  - read private data fields
  - make decisions

  . . .



- Each method must be responsible for **one** clearly defined task
- Do not define methods which "do everything" as their use will be confusing and may cause problems if you need to make some changes

Example: `double b = Math.cos(f); // calculates cos`

Would this method be generic and simple to use if besides calculating `cos` *it would prompt you to enter a value for* `f` *and then start playing music when the result is returned?*

One method – One task

Can methods receive input data?

- To complete their tasks, methods may need input data
- Input data can be passed from other methods: parameters (arguments)

```
class Example  {
   public static void main(String[] args)
   {
        String message = "Hello";

        System.out.println( message );

        double  a = 4.0,  b = 3.0, result;

        result = Math.exp( a, b );
        result = Math.exp( a, 4.0 );
        result = Math.exp( 5.5, 2.0 );
   }
}
```

*main() is a **calling method** in this example*

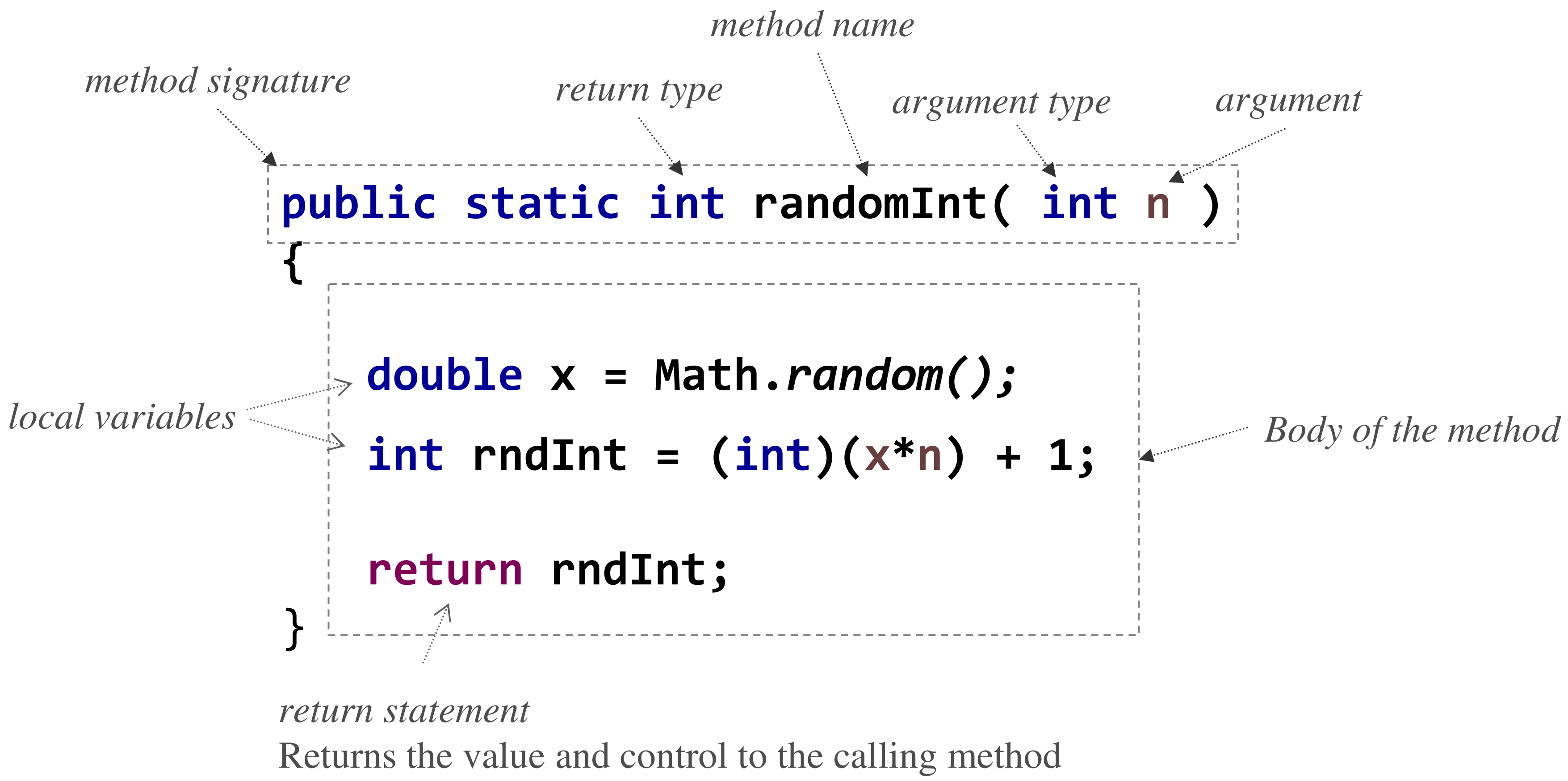
One input parameter is passed
Nothing needs to be returned back

Two input parameters are passed

One value is returned back

- If a method needs two parameters, two parameters must be passed
- Methods can also return a value back to the calling method

Method definition



```
public static int randomInt( int n )
{
    double x = Math.random();
    int rndInt = (int)(x*n) + 1;

    return rndInt;
}
```

Quiz: Where is an arithmetic expression in this method?
Explain how it is evaluated

Method definition

Generic syntax

Method signature

Method body

```
/*  Method introductory comments */

return_type methodName( formal parameter list )
{
    // local variable declarations
    // executable statements
    .  .  .  .
    return (result);
}
```

If *return_type* is void, the return statement is usually omitted

UNIVERSITY OF WOLLONGONG

Method return type

```
return_type methodName( formal parameter list )
{
        .   .    .   .    .
}
```

- If a method doesn't return anything , use `void` as its return type
- The actual type of the returned value must match the type specified in the signature
- A `return` statement can be omitted if return type is `void`
  (the method doesn't return anything)

```
short firstMt(…)
{
   short x;
   x =   . . .
   return x;
}
```

```
double secondMt(…)
{
   double time;
   time = . . .
   return (2.0*time);
}
```

```
void thirdMt(…)
{
    . . .
    . . .
}
```

Formal parameter list

```
return_type methodName( formal parameter list )
{
        .   .   .
}
```

- The list contains parameters that will be assigned with actual values (passed to the method when it is called).
  Parameters in the list are separated by commas.
- Each parameter must have a data type
- If no parameters are passed to the method, use empty brackets ()

Examples:

```
   public int calcAverage( int x1, int x2 )
   private double getHeight( double velocity, double theta )
   public void prompt( )
```

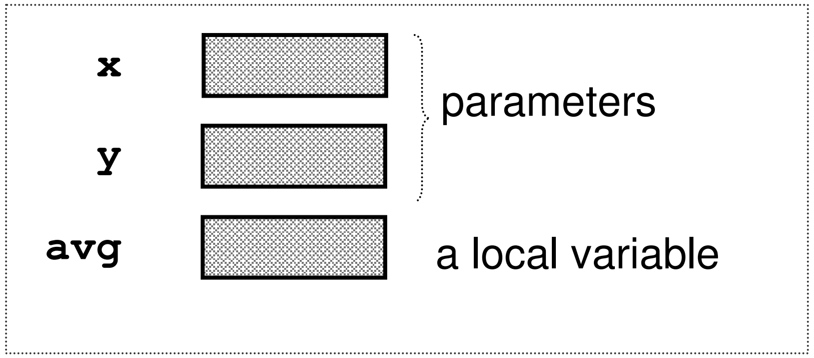
Parameters and local variables

- Local variables of a method are variables declared inside the method
- Method parameters and local variables can be used only in the executable statements **inside** the method
- Method parameters and local variables must have unique identifiers

A method

```
public int average( int x, int y )
{
        int avg;
        avg = ( x + y )/2;
        return avg;
}
```

Memory space of the method



They "exist" only inside this method
They can be used only inside this method

Quiz

- Find errors in the following definitions

```
public double multiply( int x,     y)
{
     int z, x;

     z = x*y + z;

     return z;
}
```

```
public void getAverage( double x )
{
      double sum = 0;

      sum += x;

      return (sum/2.0);
}
```

Formal and Actual parameters

```
public static void main(String[] args)
{
  double price1 = 10.0, price2 = 20.0;
  double avgPrice;
  /* Method call */
  avgPrice = findAverage( price1 , price2);
    . . . . .
}
```

Actual parameters
price1 and price2

10.0 **20.0**

Copies (rvalues) of
price1 and price2

15.00

```
   /* method definition */
 static double findAverage(double num1, double num2)
 {
     return ( (num1 + num2)/2.0 );
 }
```

Formal parameters
num1 and num2

15.00

1. Values of price1 and price2 are copied and the copies are passed to the method
2. Lvalues num1 and num2 are assigned with the passed rvalues
3. The result is returned back to main() where it is assigned to lvalue avgPrice

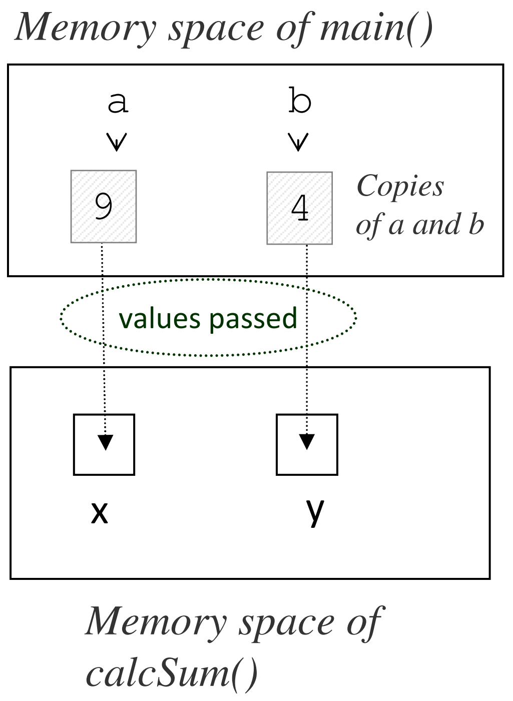
Quiz

What are the values of a and b when the method is called?

```
import static java.lang,System.*;


... main(...)
{
  int a=9, b=4;

  c = calcSum( a, b );
  return 0;
}
```

```
int calcSum( int x, int y )
{
  int sum = x + y;
  return sum;
}
```

What are the values of x and y when control is passed to this method ?



Scope

This variable (field) is visible everywhere within this class. It has a class scope

These variables `a,b,c` are local to `main()`. They have this method scope

These variables `a` and `b` are local to `calc()`. They have this method scope

Class scope

main() method scope

calc() method scope

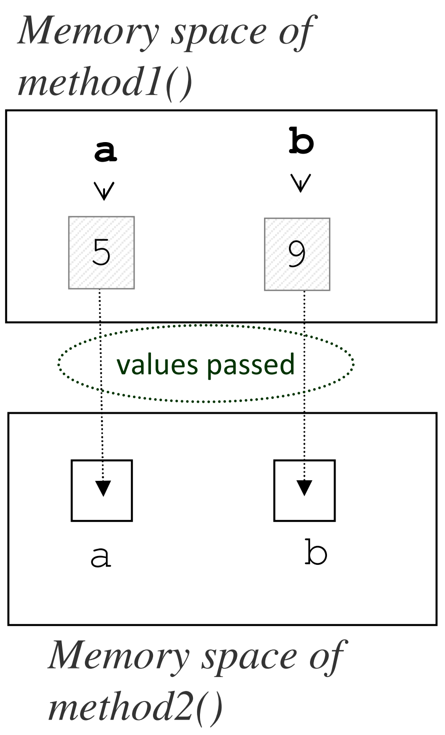
```
class Example
{

   // field are visible to all methods of the class
    private static int cV;


  public static void main(String[] args)
  {
     int a=0, b=1, c=0;
      ……
     cV = 1;   //cV is visible everywhere within the class
     c = calc(a, b); // local a,b,c are visible only
                     // inside the main() method
      ……
  }

  public static int calc(int j, int k)  {

      int a=0, b = 5;
      cV = a; //cV is visible everywhere within the class
      return ( j+k/b );   // formal parameters j, k
                          // and local variables a, b
                          // are visible only localy
  }
}
```

Quiz

Will the compiler complain about conflicting identifiers `a` and `b`?

```
public int method1()
{
   int a=5, b=9;

   int result = method2( a, b );
   return 0;
}
```

```
public int method2( int a, int b )
{
   int result = a + b;
   return result/2;
}
```



Identical names

- Local variables of one method cannot have conflicting names with variables defined in other methods (their scope is limited to a method)
- What if a local variable has the same name as one of the data fields (a class scope variable) ?

```
class Example
{           Why static
   private static int number, code;

   public static void main(String[] args) {
       number = 1;  // the class scope variable is used
       testLocal();
       out.println( "main() method: " + number + " " + code);
   }       Why static
   public static void testLocal() {
       int number = 99;     // a method scope variable is declared (local)

       out.println( "testLocal() method: " + number + " " + code);
   }
}
```

If a local variable has the same name as a class scope variable, the local one has a higher priority

Calling methods (within a class)

- Once a method is defined in a class, it can be called directly from other methods in the same class

```
class Example
{        // method definition
      public int getSum(int n1, int n2) {
         return n1+n1 ;
      }

      public void printResult() {
         int sum = getSum( 2, 17 ); // a call within a class
         .  .  .
      }
}
```

$$\begin{tikzcd} \text{getSum} & \text{17} \\ \textit{method name} \arrow[u, dotted] & \textit{actual parameters} \arrow[u, dotted] \end{tikzcd}$$

1. Specify the method name **`getSum`**
2. Provide actual input parameters … **`( 2, 17 )`**
3. Assign to a variable that matches the type returned by the method
   `int sum = … ;` *(only if the method returns a value)*

Calling static methods

- Static methods defined in a class as **`public`** can be called from other classes through the class name

```
class Example
{       // method definition
        public static void printNumber( int n1 ) {
            System.out.println(n1);
        }
}
```

```
class Test {
        public void printResult() {

            Example.printNumber( 12 );     // a call from another class
        }
}
```

(1) class name   the dot   (2) method name   (3) actual parameter

1. Specify the class name
   (*another example*: **`Math`**`.cos( angle )`)
1. Specify the method name
2. Provide actual input parameters

Calling methods

- Methods defined in a class as **`public`** can be called from other classes only through a reference variable

```
class Example
{     // method definition
      public void printNumber(int n1) {
         System.out.println(n1);
      }
}
```

```
class Test {
   public void printResult() {
      Example exVar = new Example(); // an object must be declared
      exVar.printNumber( 12 );      // a call from another class
   }
}
```

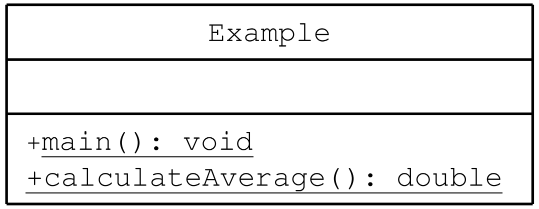
(1) object name *the dot* *(2) method name* *(3) actual parameters*

1. Declare an object
2. Specify the object name
3. Specify the method name
4. Provide actual input parameters

Defining a class with two methods

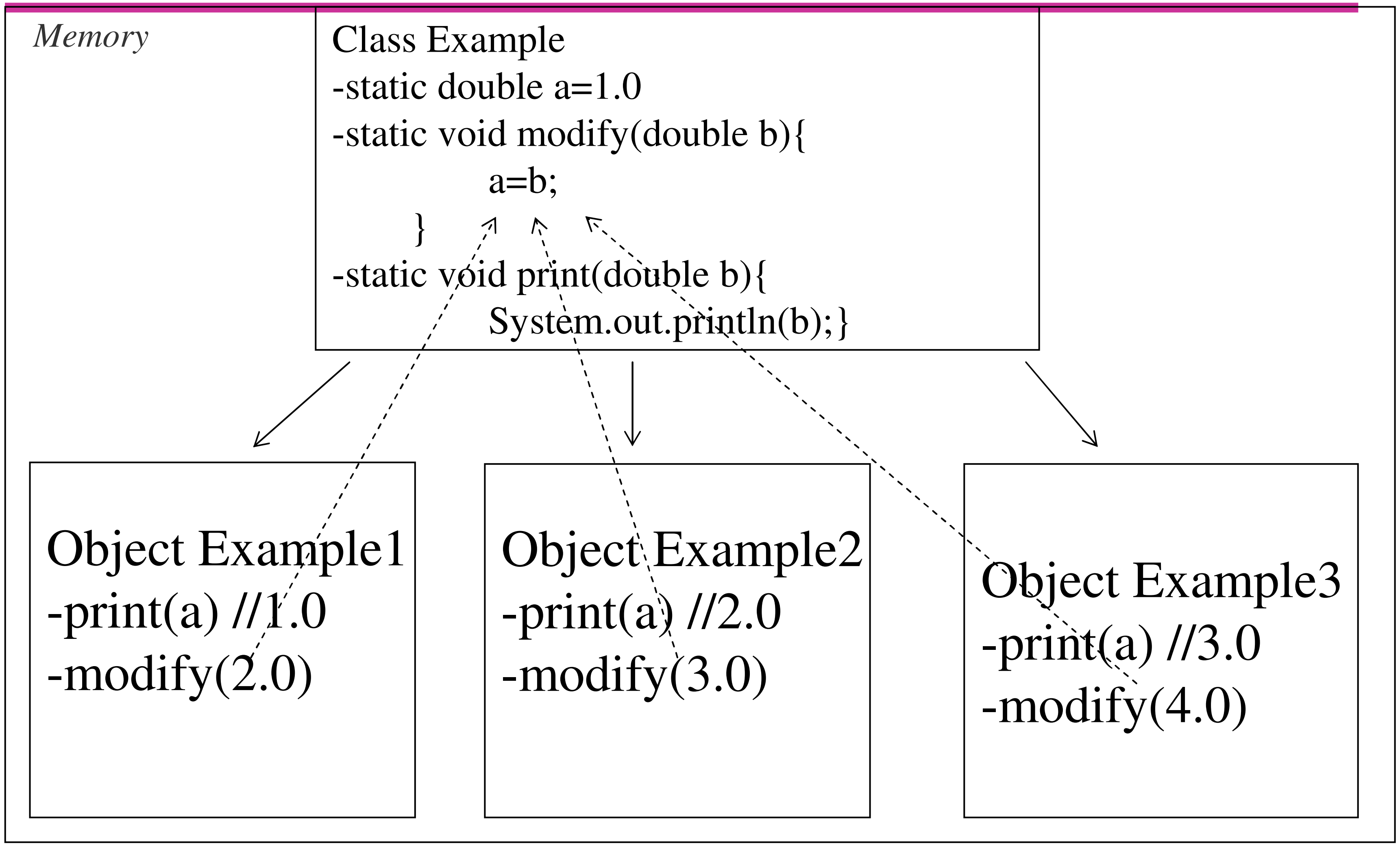
Example:

1. main() - input two double numbers separated by space
2. calculateAverage() is called from main() to calculate the average
3. main() – print the result returned by calculateAverage()



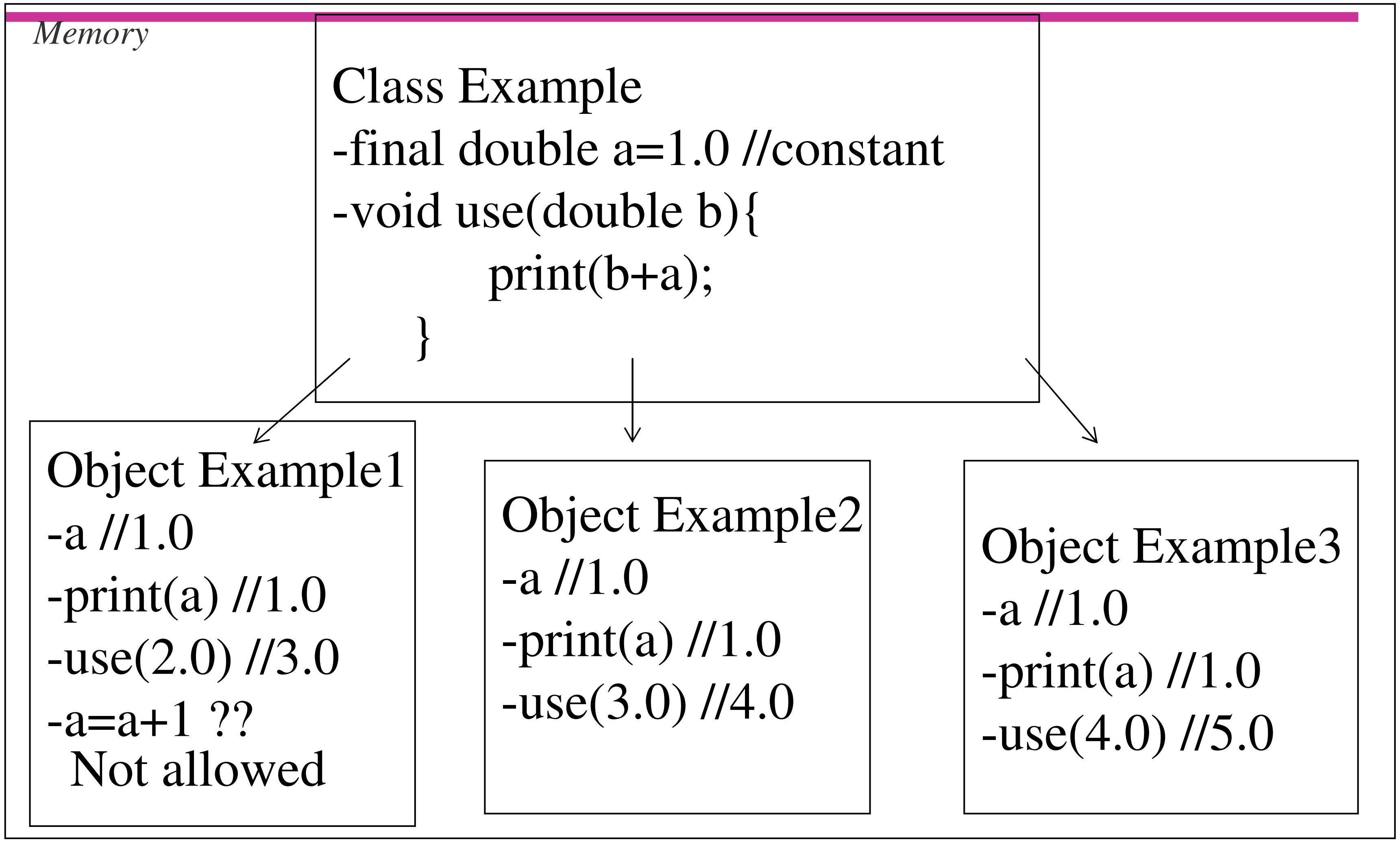
```
class Example  {
    public static void main(String[] args)  {
        Scanner inp = new Scanner(System.in);

        System.out.print("Input two numbers: ");
        double d1 = inp.nextDouble();
        double d2 = inp.nextDouble();

        double average = calculateAverage( d1, d2 ); // method call

        System.out.println("The average is :" + average);
    }
          Why static
    public static double calculateAverage(double v1, double v2)  {
        double average = (v1 + v2)/2.0;
        return average; // this is a locally declared average
    }
}
```

Static vs. Constant



Objects share the same value in memory. Modifications in objects are allowed, and the result will automatically affect all other objects of the same class.

Static vs. Constant



Class Example
-final double a=1.0 //constant
-void use(double b){
            print(b+a);
     }

Object Example1
-a //1.0
-print(a) //1.0
-use(2.0) //3.0
-a=a+1 ??
  Not allowed

Object Example2
-a //1.0
-print(a) //1.0
-use(3.0) //4.0

Object Example3
-a //1.0
-print(a) //1.0
-use(4.0) //5.0

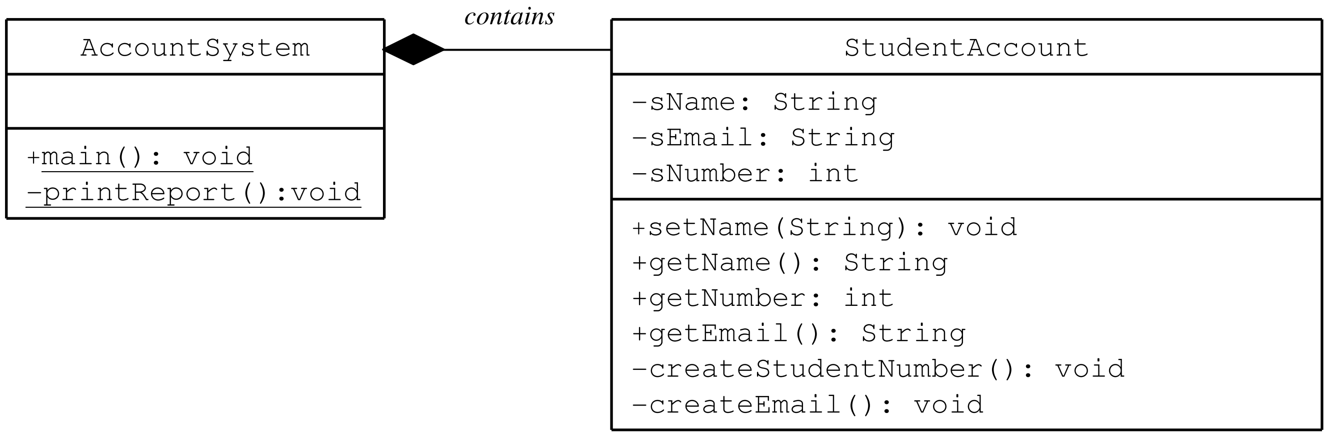
Objects keep different instances in memory, but modifications in objects are NOT allowed

A program with two classes

- You can add more methods and data fields to the class **`Example`** However, in order to be used with the static **`main()`** method, all of them should be static too
- It doesn't make sense to create objects of a class if **all** its fields are static: all instances will have exactly the same state. Their interaction is useless

Solution 1:

Define another class with non-static methods and fields using composition



A program with two classes

- It is not a good practice to implement all program components and then compile and debug it
  - you are very likely to end up with many compilation errors
  - when you fix one bug there is a chance that you'll introduce a new one

Common solution: Incremental development combined with debugging
  - implement the simples possible version that can be compiled
  - add elements one-by-one, compile, fix bugs

It's possible (simple to start with) to define two classes in one .java file

File name: AccountSystem.java

```
class StudentAccount
{


}


class AccountSystem
{
    public static void main(String[] args)
    {
    }
}
```

A program with two classes

- More details added

The field sName has not been initialised

File name: AccountSystem.java

```
class StudentAccount
{
    private String sName; // a private field that can be accessed via
                          //  public methods defined in this class
    public String getName() { return sName; }
}

class AccountSystem
{
    public static void main(String[] args)
    {
        StudentAccount stud1 = new StudentAccount(); // create an object
        String name1 = stud1.getName(); //call a public method to get sName
        System.out.println("Student's name: " + name1);
    }
}
```

```
Student's name: null
```

A program with two classes

File name: AccountSystem.java

```
import java.util.Scanner;
class StudentAccount
{
    private String sName;
    public String getName() { return sName; }
    public void setName(String name) { sName = name; }
}

class AccountSystem
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner( System.in );
        System.out.print(" Enter a name: ");
        String aName = keyboard.next(); // read a word
        StudentAccount stud1 = new StudentAccount(); // create an object
        stud1.setName( aName );  // call a public method to set sName
        String name1 = stud1.getName(); // get a private field
        System.out.println("Student's name: " + name1);
    }
}
```

this reference variable

- What if a formal parameter of a method has the same identifier as a class scope variable?

```
class StudentAccount
{
    private String name;
    .   .   .
    public void setName(String name) {
        name = name;   // which name?
    }
}
```

As the local scope has a higher priority than the class scope, the formal parameter `name` *will be assigned to itself*

Solution 1: Use different names
Solution 2: Use a special **`this`** reference variable
`this.variable` is always referencing a class scope variable

```
class StudentAccount   {
    private String name;
      .   .   .
    public void setName(String name) {
        this.name = name;
    }
}
```

A program with two classes

- As the source file grows it may be more convenient to implement each class in a separate file
- To compile both files together:

  `javac StudentAccount.java AccountSystem.java`

File: StudentAccount.java

```java
class StudentAccount
{
    private String sName;
    public String getName() { return sName; }
    public void setName(String name) { sName = name; }
}
```

File: AccountSystem.java

```java
import java.util.Scanner;

class AccountSystem
{
    public static void main(String[] args)
    {
          Scanner keyboard = new Scanner( System.in );
          System.out.print(" Enter a name: ");
          String aName = keyboard.next(); // read a word from the input buffer
          StudentAccount stud1 = new StudentAccount(); // create an object
          stud1.setName( aName );  // call a public method to set private field sName
          String name1 = stud1.getName(); // get a private field via a public method
          System.out.println("Student's name: " + name1);
    }
}
```

Suggested reading

Java: How to Program (Early Objects), 10th Edition

- Chapter 3: Introduction to Classes