Object Oriented Programming

Dr. D. Sudheer,
Asst. Prof. Sr.
SCOPE
VIT-AP

Learning Objectives

- To study clearly about classes and objects.
- To assign object reference variable.
- To introduce methods.
- To study of constructors.
- To introduce this keyword.

Java Classes

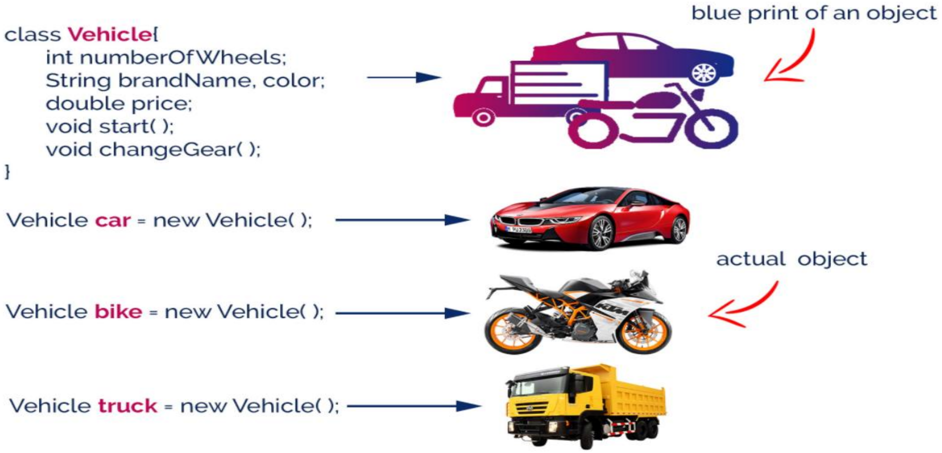
- Java is an object-oriented programming language.
- everything in java program must be based on the object concept.
- In a java programming language, the class concept defines the skeleton of an object.
- The java class is a template of an object.
- Once a class got created, we can generate as many objects as we want.
- All the objects of a class have the same properties and behaviors that were defined in the class.

Every class of java programming language has the following characteristics:

Identity - It is the name given to the class.

State - Represents data values that are associated with an object.

Behavior - Represents actions can be performed by an object.



```
class Vehicle{
     int numberOfWheels;
     String brandName, color;
     double price;
     void start( );
     void changeGear( );
}

Vehicle car = new Vehicle( );

Vehicle bike = new Vehicle( );

Vehicle truck = new Vehicle( );
```

Java Constructors

- A constructor in Java is a **special method** that is used to initialize objects.
- The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:
- Characteristics of constructors:
- The name of the constructor must be same as that of the class
- No return type can be specified for constructor
- A constructor can have parameter list
- The constructor function can be overloaded
- They cannot be inherited but a derived class can call the base class constructor
- The compiler generates a constructor, in the absence of a user defined constructor.
- Compiler generated constructor is public member function
- The constructor is executed automatically when the object is created
- A constructor can be used explicitly to create new object of its class type

Java Constructors

Default Constructor

- A default constructor is a **0 argument constructor** which contains a no-argument call to the super class constructor.
- To assign default values to the newly created objects is the main responsibility of default constructor.
- Compiler writes a default constructor in the code only if the program does not write any constructor in the class.
- The access modifier of default constructor is always the same as a class modifier but this rule is applicable only for **“public”** and **“default”** modifiers.

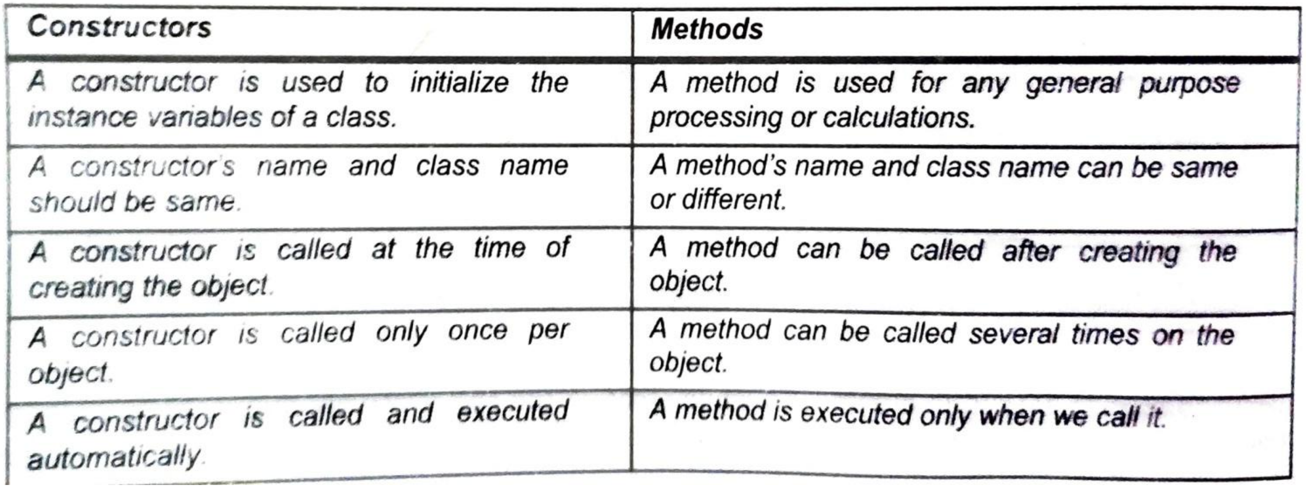
When will compiler add a default constructor

- The compiler adds a default constructor to the code only when the programmer writes no constructor in the code.
- If the programmer writes any constructor in the code, then the compiler doesn't add any constructor.
- Every default constructor is a 0 argument constructor but every 0 argument constructor is not a default constructor.

Java Constructors

Parameterized Constructors

- The parameterized constructors are the constructors having a **specific number of arguments** to be passed.
- The purpose of a parameterized constructor is to assign user-wanted specific values to the instance variables of different objects.
- A parameterized constructor is written explicitly by a programmer.
- The access modifier of default constructor is always the same as a class modifier but this rule is applicable only for **“public”** and **“default”** modifiers.



Constructors	Methods
A constructor is used to initialize the instance variables of a class.	A method is used for any general purpose processing or calculations.
A constructor's name and class name should be same.	A method's name and class name can be same or different.
A constructor is called at the time of creating the object.	A method can be called after creating the object.
A constructor is called only once per object.	A method can be called several times on the object.
A constructor is called and executed automatically.	A method is executed only when we call it.

Java Methods

•A method is a block of statements under a name that gets executes only when it is called.

•Every method is used to perform a specific task.

•The major advantage of methods is code re-usability

•Every method in java must be declared inside a class.

•Every method declaration has the following characteristics:

returnType - Specifies the data type of a return value.

name - Specifies a unique name to identify it.

parameters - The data values it may accept or receive.

{ } - Defines the block belongs to the method.

Java Input/Output

•Input represents data given to the program. Output represents data displayed as result.

• System.out.print() or System.out.println() are used to display output.

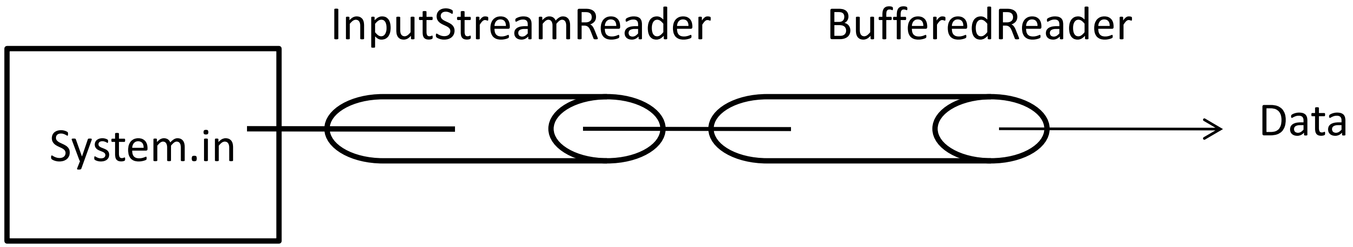
•Stream is required to accept input from user. Stream is flow of data from one place to another.

•There are two streams: 1. Inputstream 2. Outputstream.

•Input streams are used to read data from some other place.

•Output streams are used to write data from some other place.

•All streams are represents in java.io. Package.



Reading input from user

Using Scanner and Using BufferedReader

•**Scanner** is a class in **java.util** package that is used to get input from standard I/O or files with primitives types such as int, double, strings…

•The BufferedReader class of Java is used to read the stream of characters from the specified source (character-input stream).

Benefits

Using Scanner will help us parse, convert to our desired data type without implementing our self.

We can customize separator in Scanner to get what we want.

Drawbacks

Scanner has a little buffer (1KB char buffer).

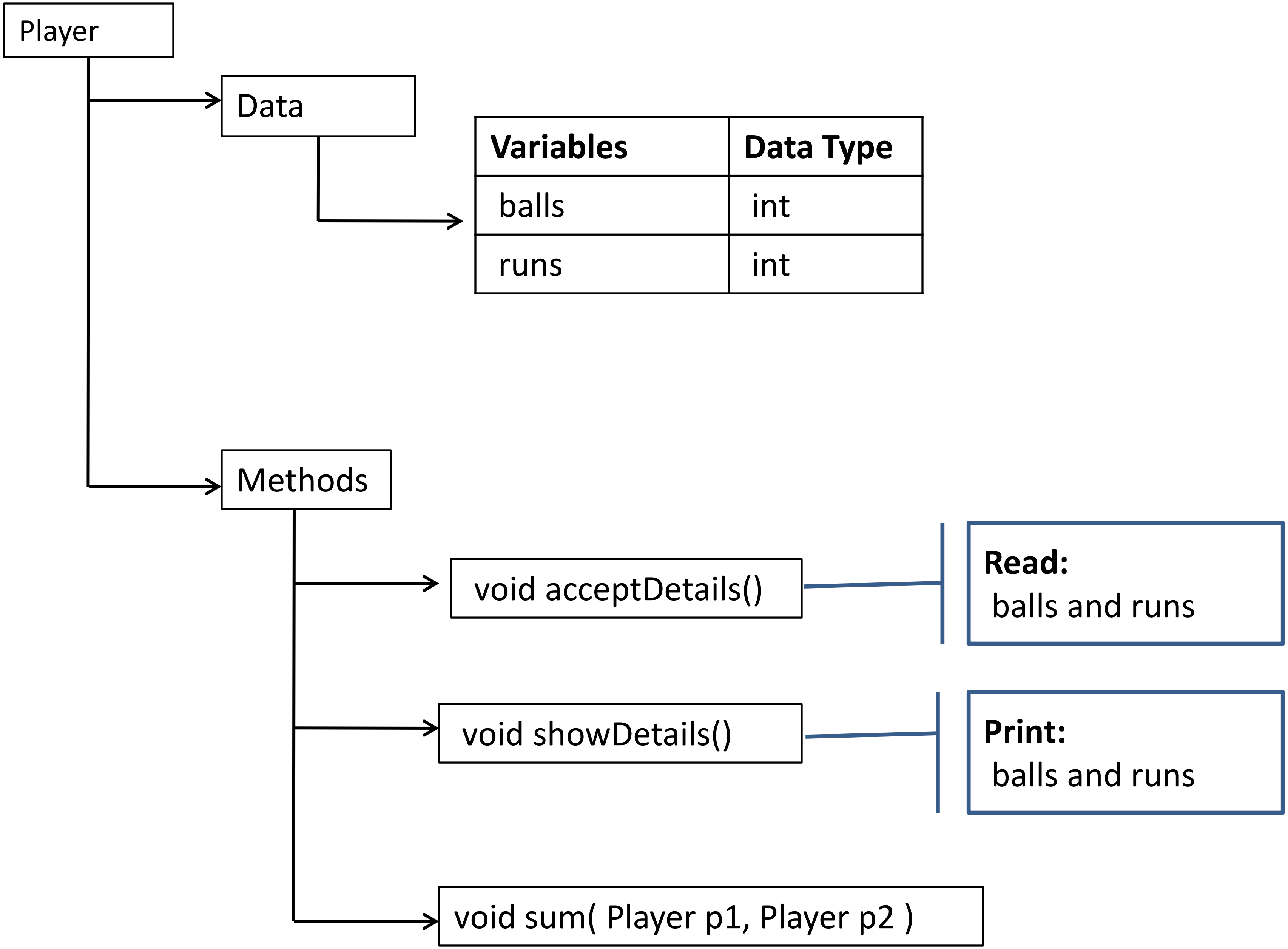
Scanner is slower than BufferedReader because Scanner does parsing of input data, and BufferedReader simply reads sequence of characters.

A Scanner is not safe for multithreaded use without external synchronization.

Enhancement of Methods:

- ❑ Passing Objects as Parameters in Methods
- ❑ Methods returning Objects
- ❑ Static Methods
- ❑ Command-line arguments in Main Method
- ❑ Methods Using Variable Arguments
- ❑ Recursive Methods

Passing objects as parameters in methods



Player

Data

Variables	Data Type
balls	int
runs	int

Methods

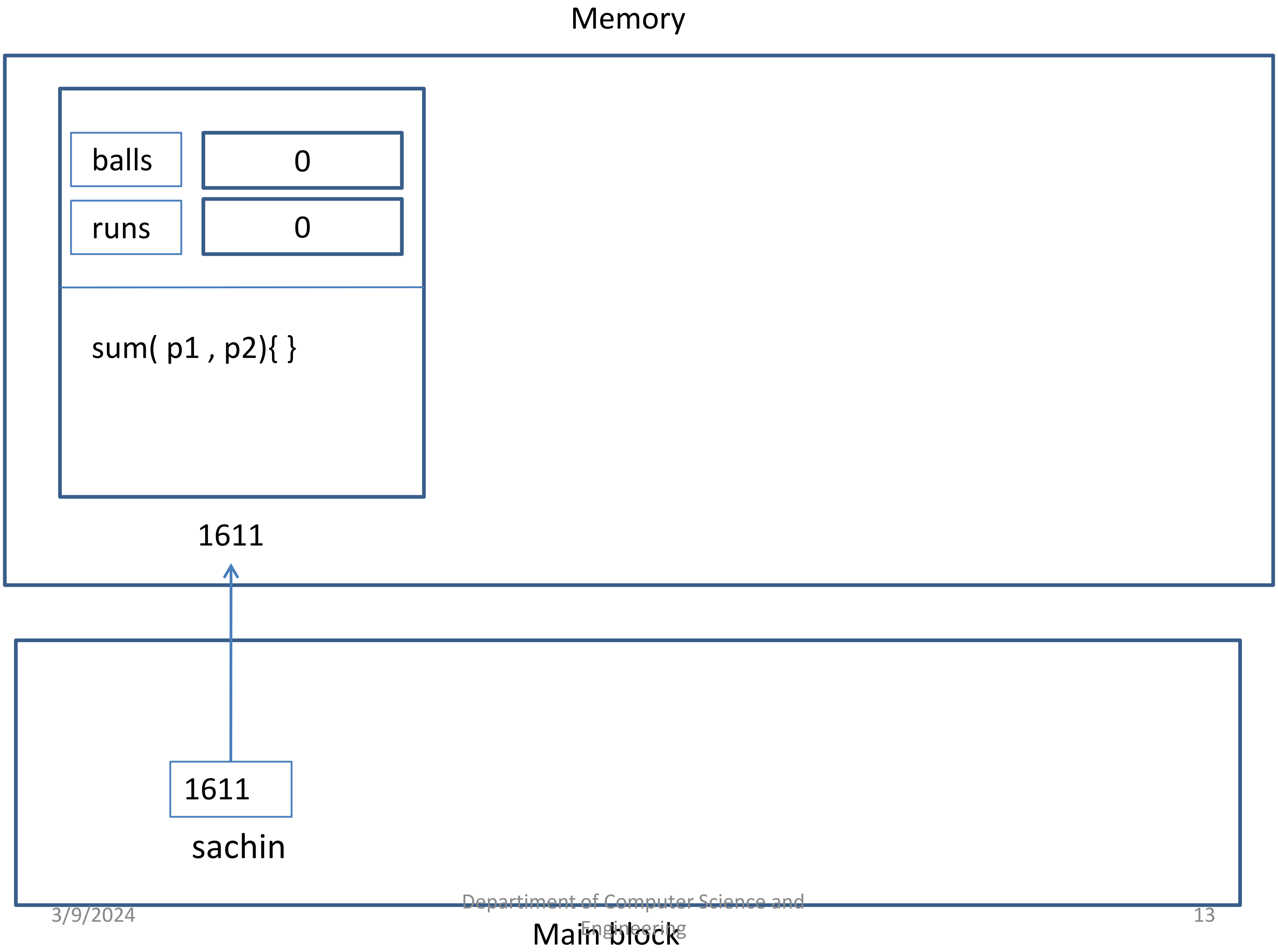
void acceptDetails()

Read:
balls and runs

void showDetails()

Print:
balls and runs

void sum(Player p1, Player p2)



Memory

balls 0

runs 0

sum(p1 , p2){ }

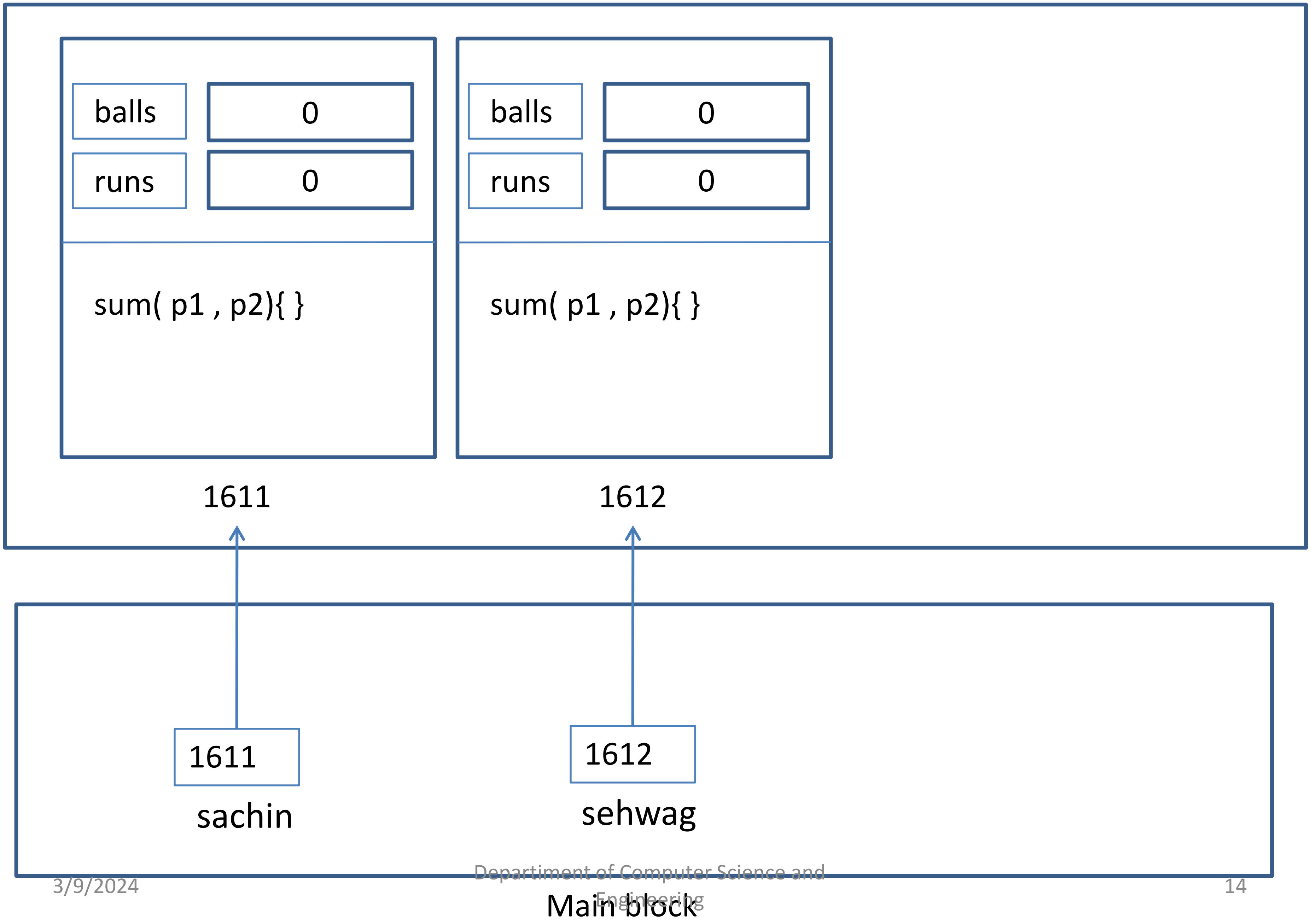
1611

1611

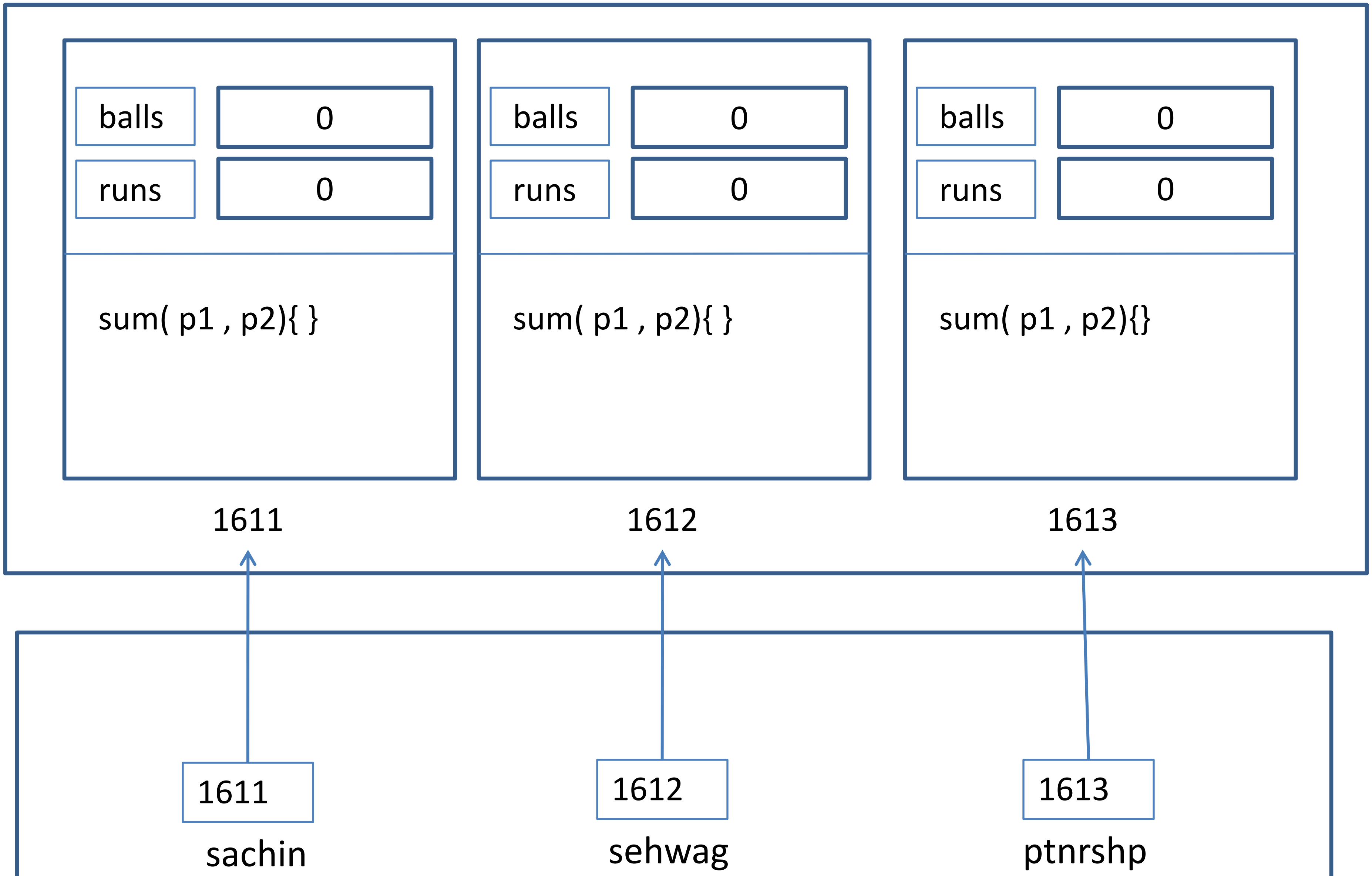
sachin

Main block

Memory


balls
0
runs
0
sum(p1 , p2){ }
1611
balls
0
runs
0
sum(p1 , p2){ }
1612
1611
sachin
1612
sehwag
Main block

Memory



balls 0
runs 0
sum(p1 , p2){ }
1611

balls 0
runs 0
sum(p1 , p2){ }
1612

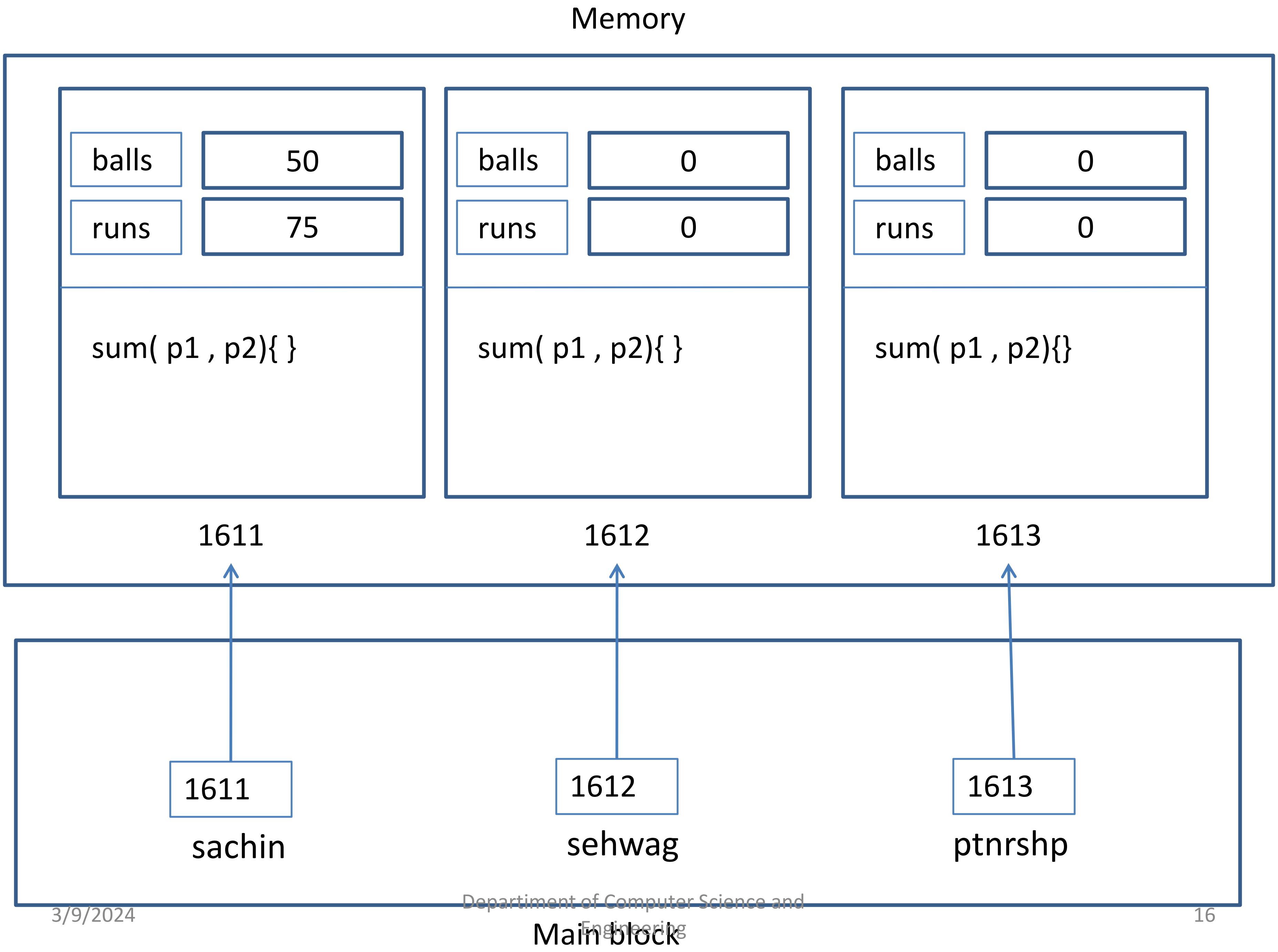
balls 0
runs 0
sum(p1 , p2){}
1613

1611
sachin

1612
sehwag

1613
ptnrshp

Main block



Memory

balls 50
runs 75
sum(p1 , p2){ }
1611

balls 0
runs 0
sum(p1 , p2){ }
1612

balls 0
runs 0
sum(p1 , p2){}
1613

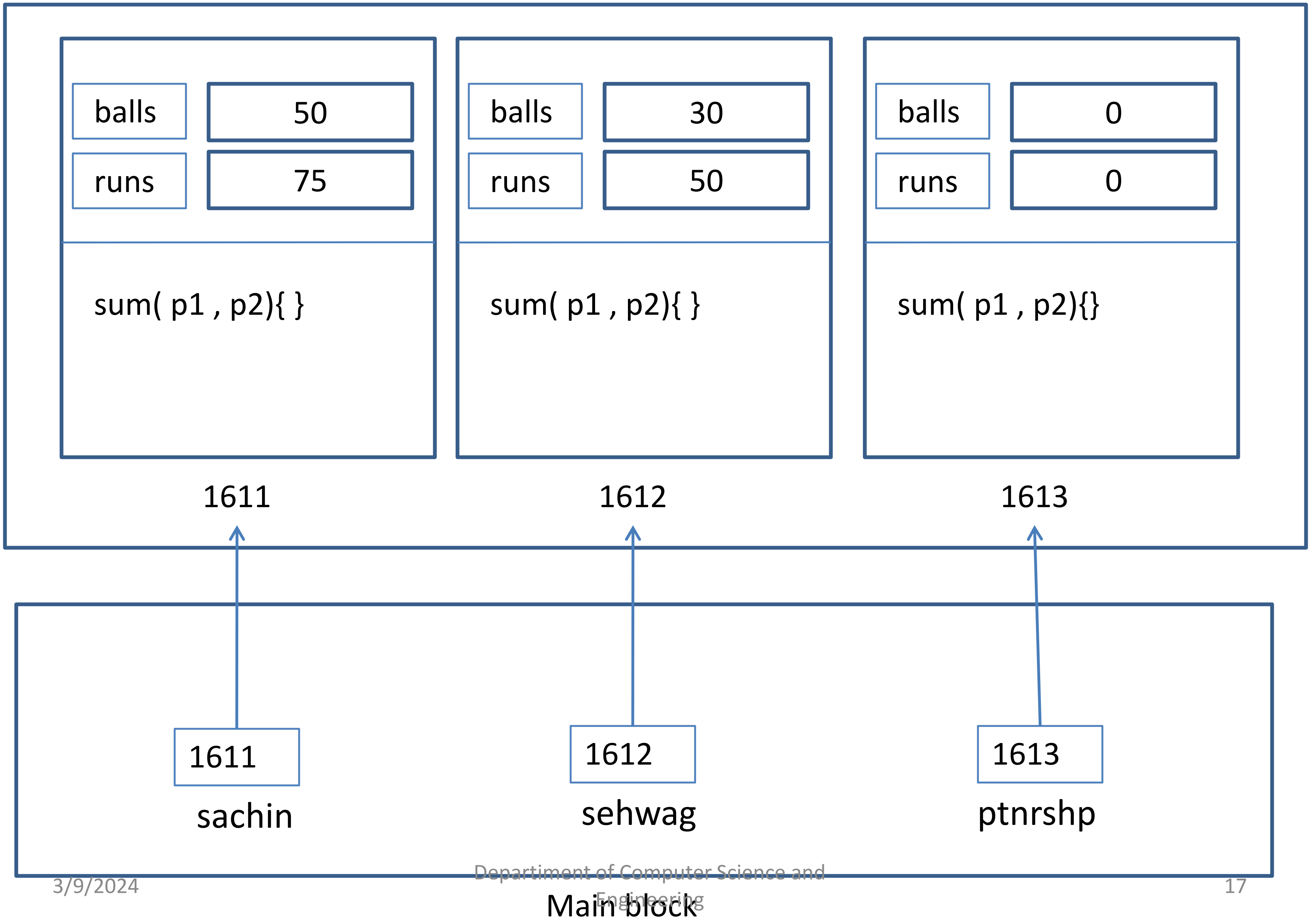
1611
sachin

1612
sehwag

1613
ptnrshp

Main block

Memory



balls 50

runs 75

sum(p1 , p2){ }

1611

balls 30

runs 50

sum(p1 , p2){ }

1612

balls 0

runs 0

sum(p1 , p2){}

1613

1611

sachin

1612

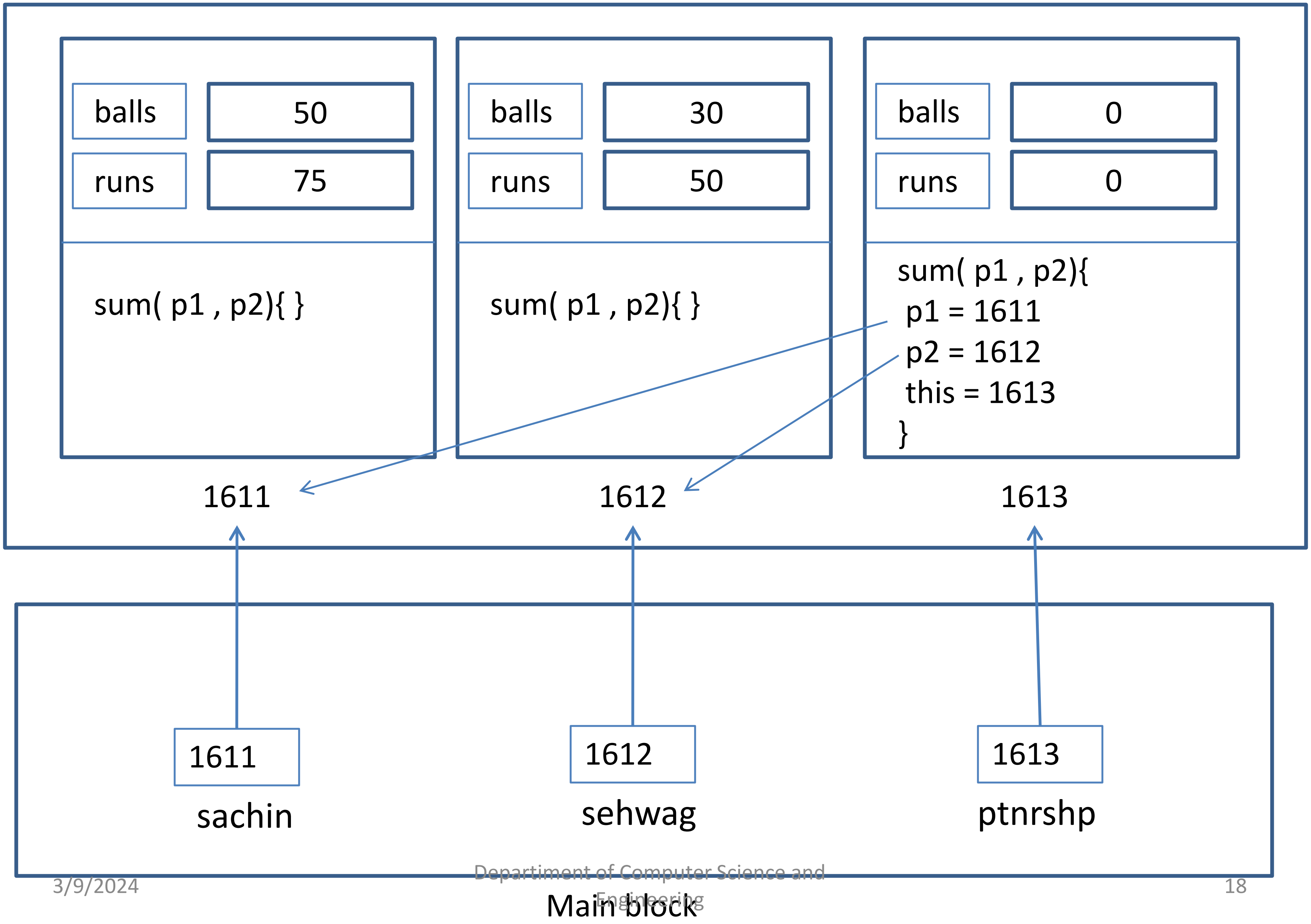
sehwag

1613

ptnrshp

Main block

Memory



balls 50

runs 75

sum(p1 , p2){ }

1611

balls 30

runs 50

sum(p1 , p2){ }

1612

balls 0

runs 0

sum(p1 , p2){
p1 = 1611
p2 = 1612
this = 1613
}

1613

1611

sachin

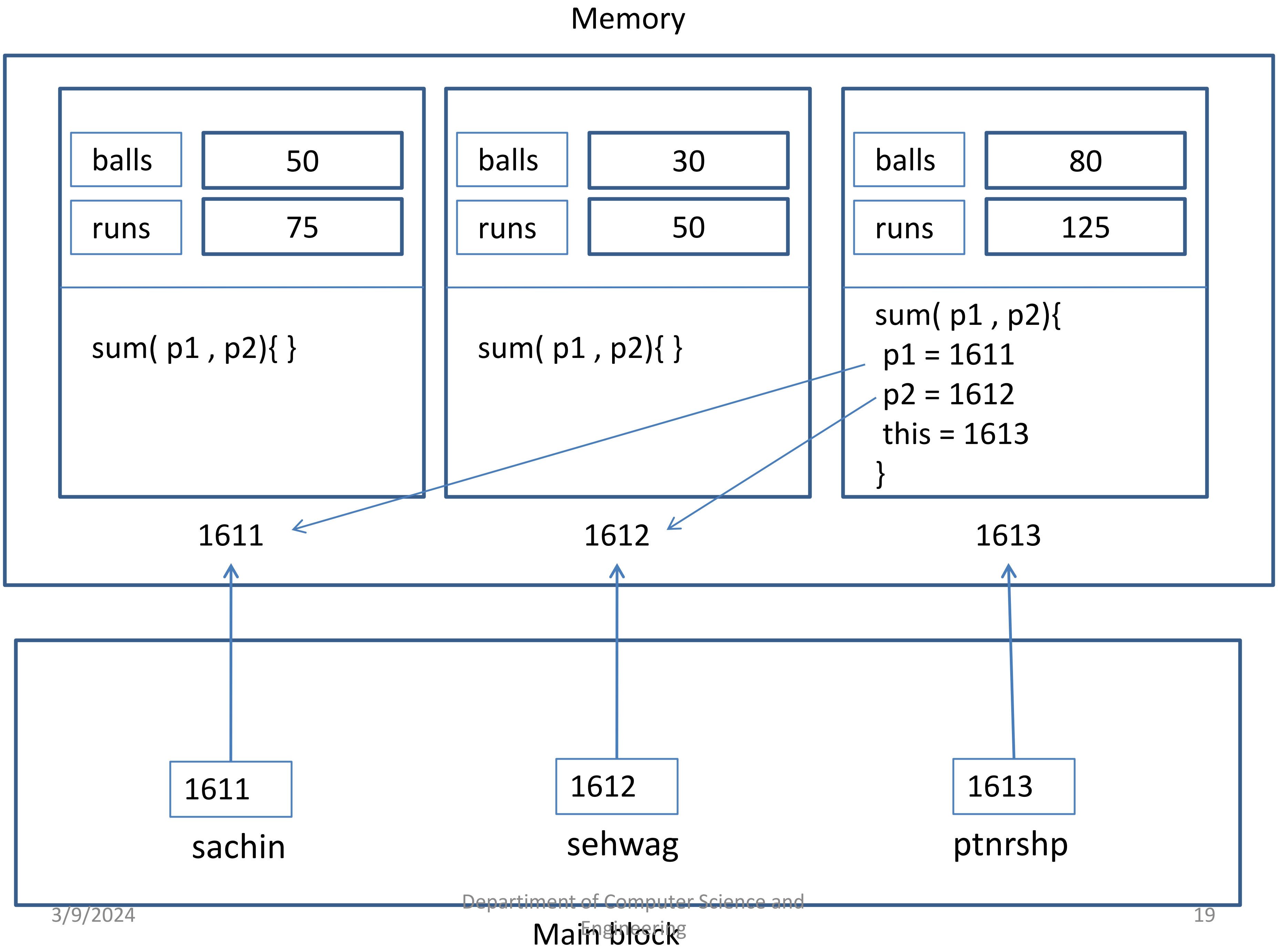
1612

sehwag

1613

ptnrshp

Main block



Memory

balls 50
runs 75
sum(p1 , p2){ }
1611

balls 30
runs 50
sum(p1 , p2){ }
1612

balls 80
runs 125
sum(p1 , p2){
p1 = 1611
p2 = 1612
this = 1613
}
1613

1611
sachin

1612
sehwag

1613
ptnrshp

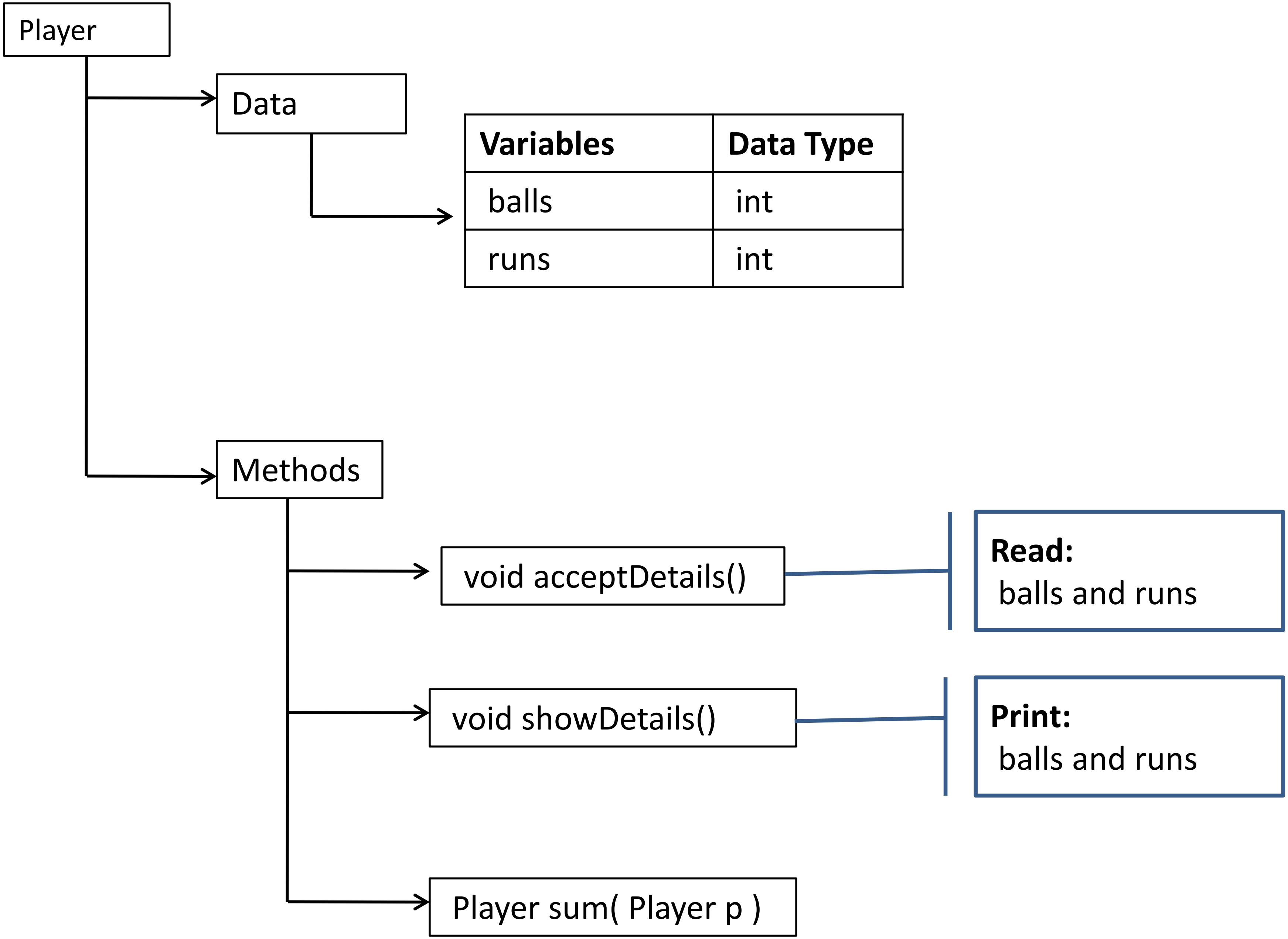
Main block

‘this’ key word

- The this keyword refers to the current object in a method or constructor.

- The most common use of the this keyword is to eliminate the confusion between instance variables and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).
- This can be used to invoke current class method.
- This can be used to invoke current class constructor.
- When object is created to class, then the default reference is also created internally named as ‘this’.

Methods returning objects



Player

Data

Variables	Data Type
balls	int
runs	int

Methods

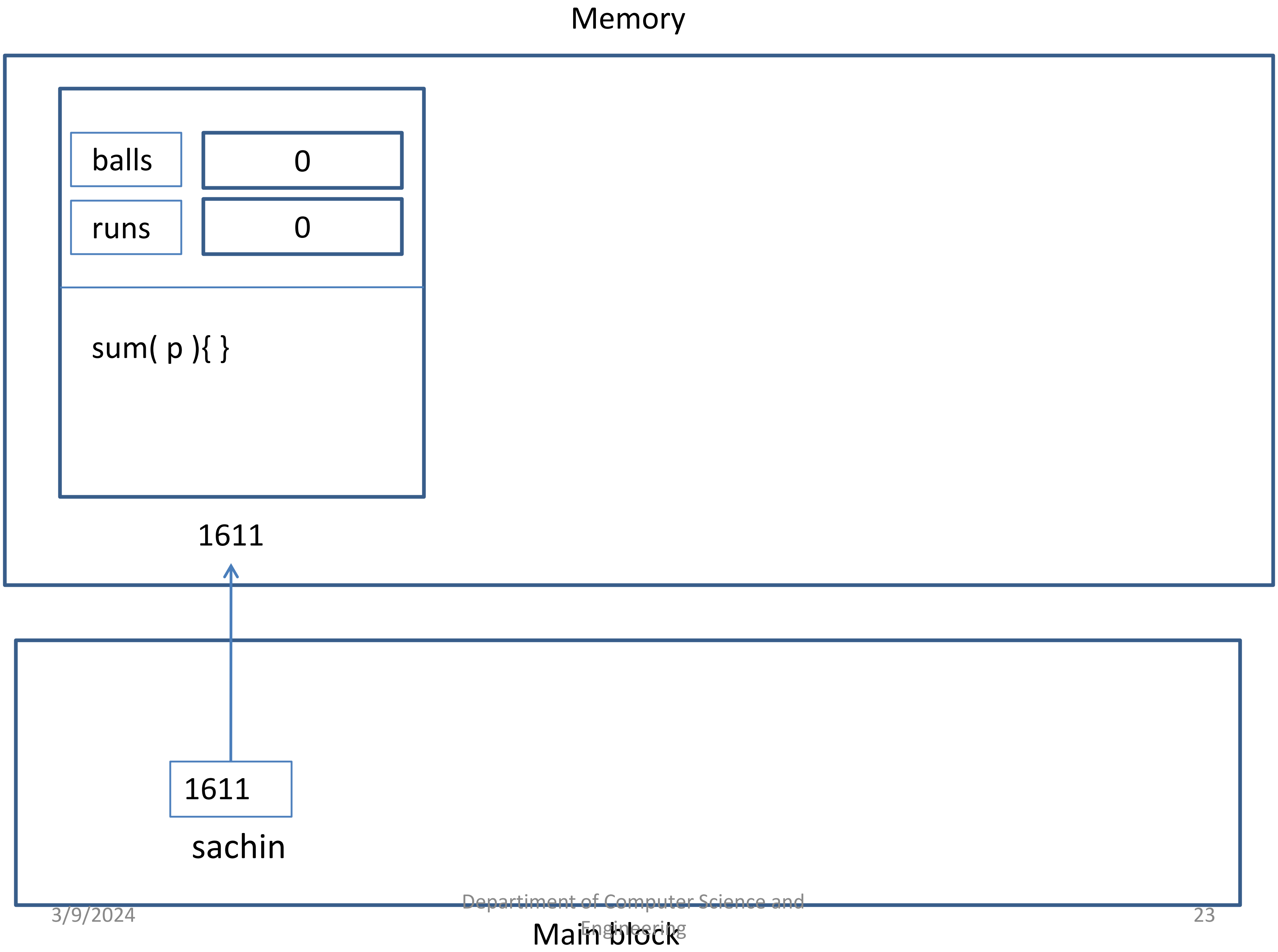
void acceptDetails()

Read:
balls and runs

void showDetails()

Print:
balls and runs

Player sum(Player p)



Memory

balls

0

runs

0

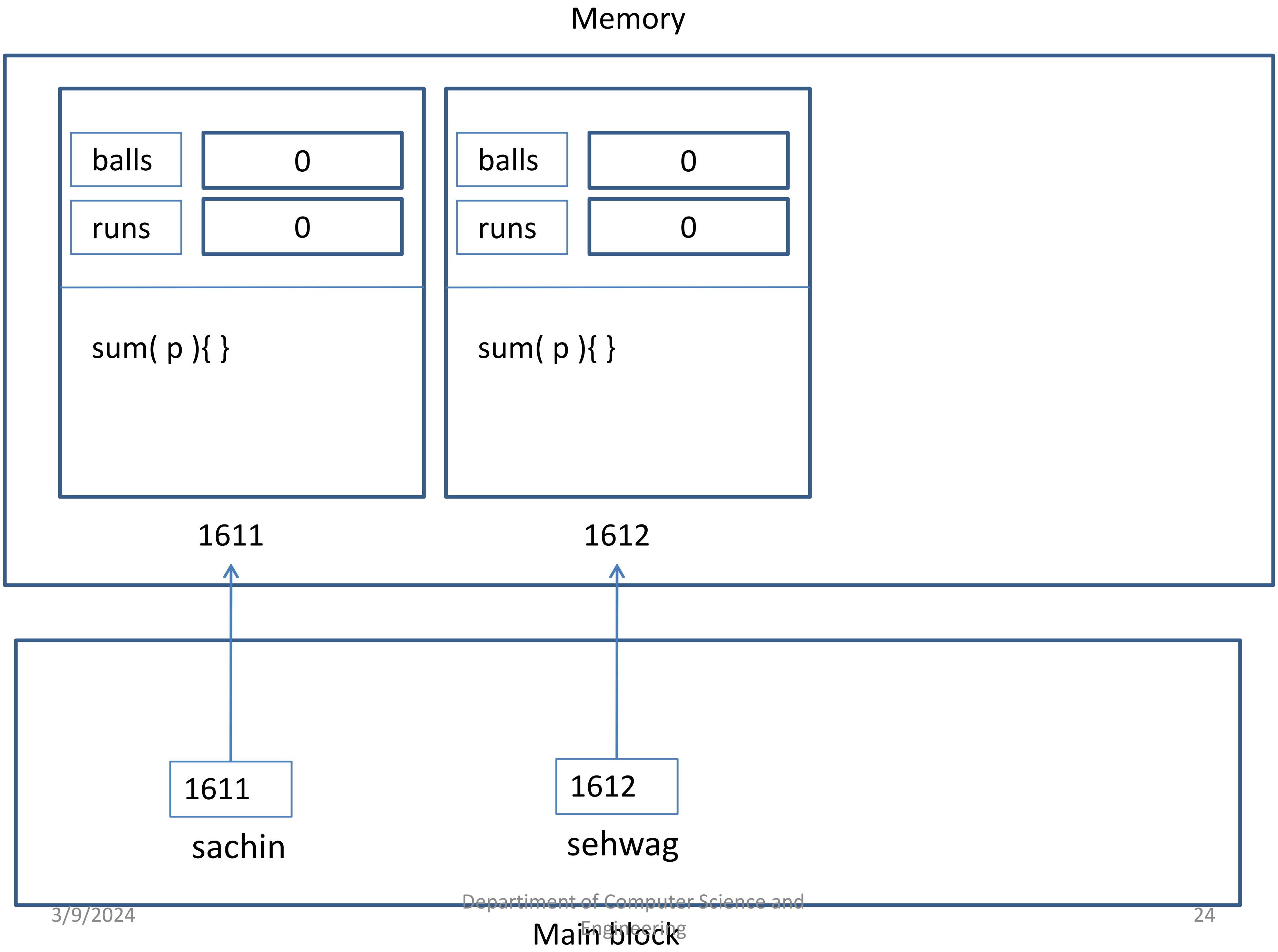
sum(p){ }

1611

1611

sachin

Main block



Memory

balls 0

runs 0

sum(p){ }

1611

balls 0

runs 0

sum(p){ }

1612

1611

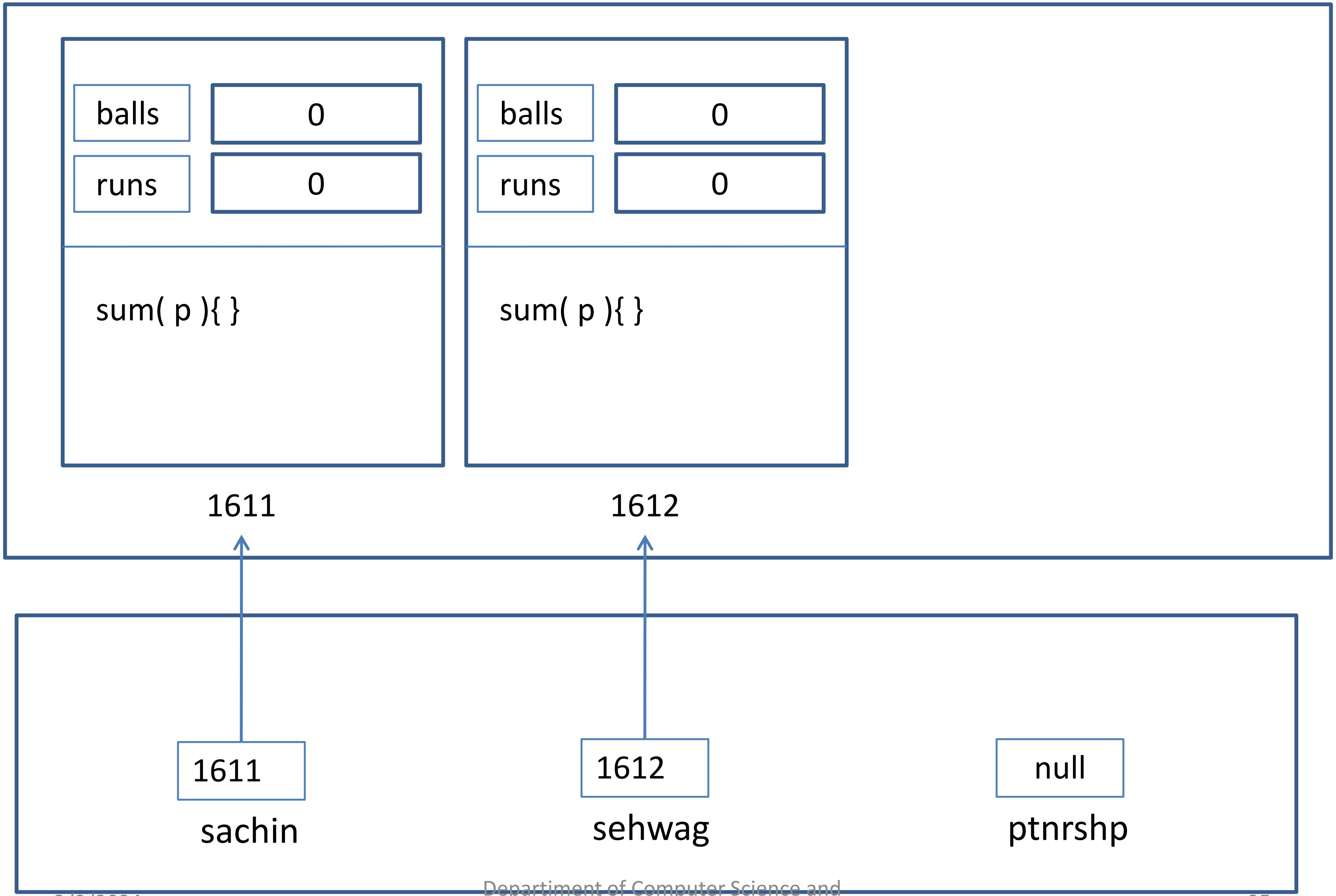
sachin

1612

sehwag

Main block

Memory



balls 0

runs 0

sum(p){ }

1611

balls 0

runs 0

sum(p){ }

1612

1611

sachin

1612

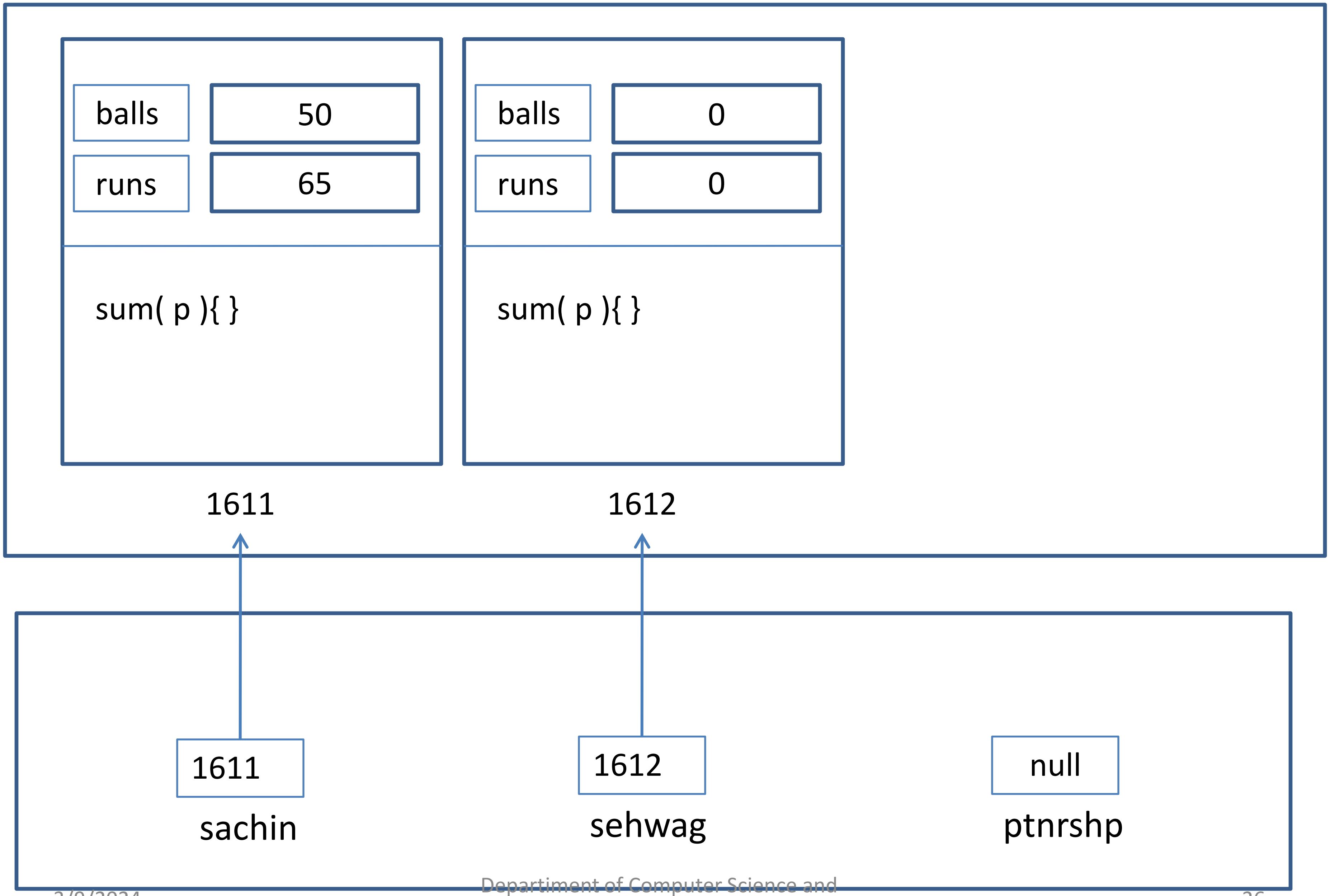
sehwag

null

ptnrshp

Main block

Memory



balls 50
runs 65
sum(p){ }
1611

balls 0
runs 0
sum(p){ }
1612

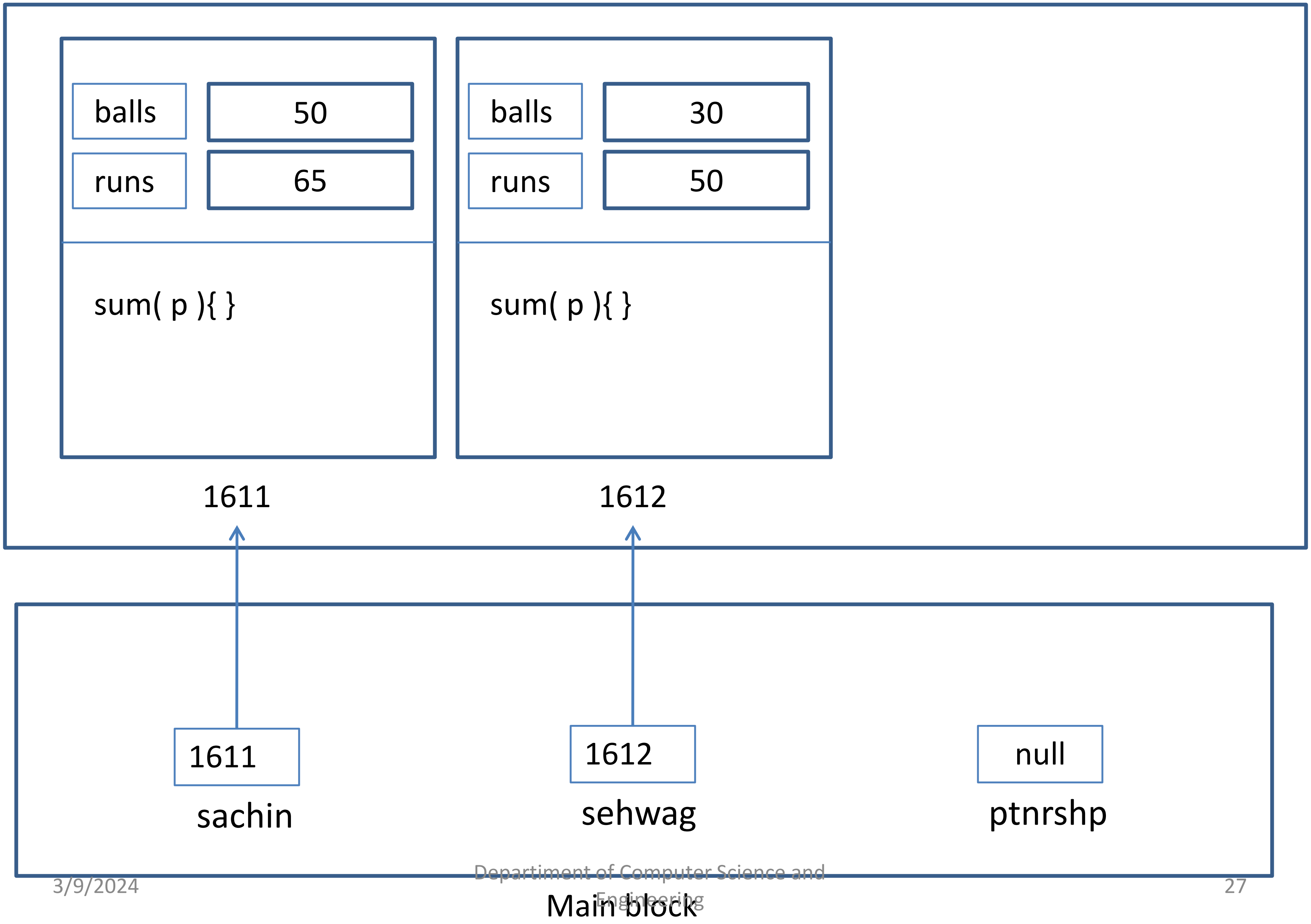
1611
sachin

1612
sehwag

null
ptnrshp

Main block

Memory



balls 50

runs 65

sum(p){ }

1611

balls 30

runs 50

sum(p){ }

1612

1611

sachin

1612

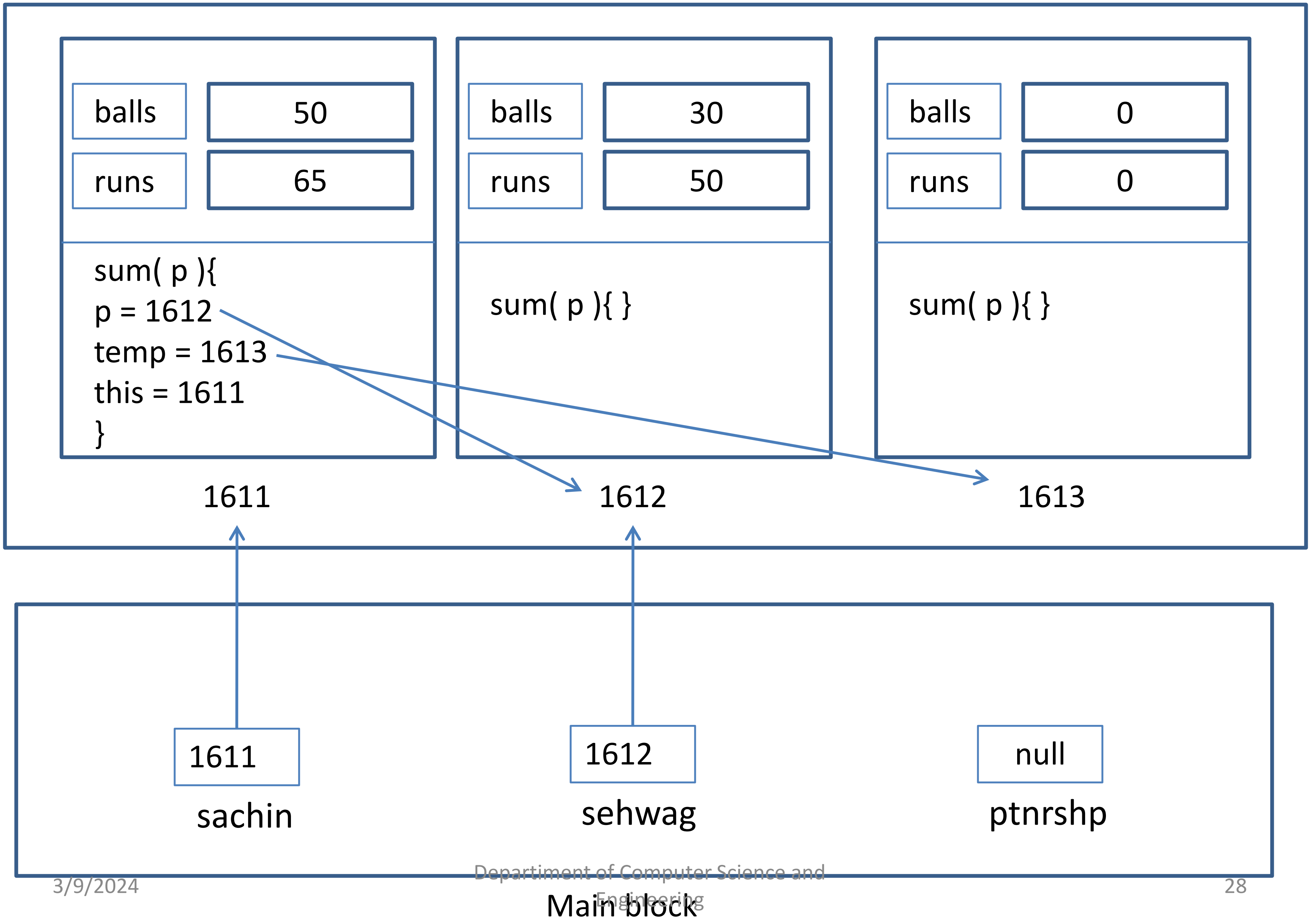
sehwag

null

ptnrshp

Main block

Memory



balls 50
runs 65

sum(p){
p = 1612
temp = 1613
this = 1611
}

1611

balls 30
runs 50

sum(p){ }

1612

balls 0
runs 0

sum(p){ }

1613

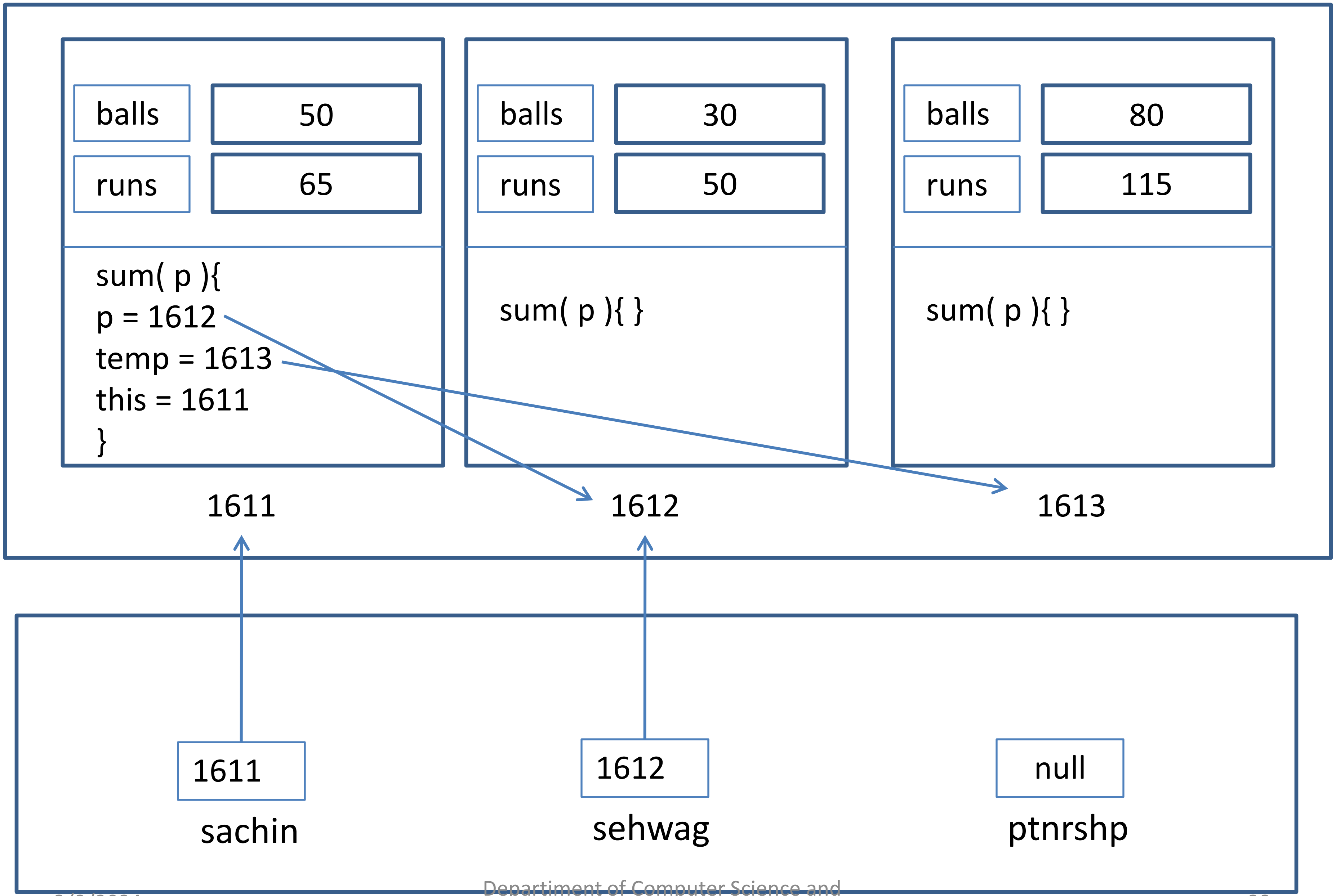
1611
sachin

1612
sehwag

null
ptnrshp

Main block

Memory



balls 50
runs 65
sum(p){
p = 1612
temp = 1613
this = 1611
}
1611

balls 30
runs 50
sum(p){ }
1612

balls 80
runs 115
sum(p){ }
1613

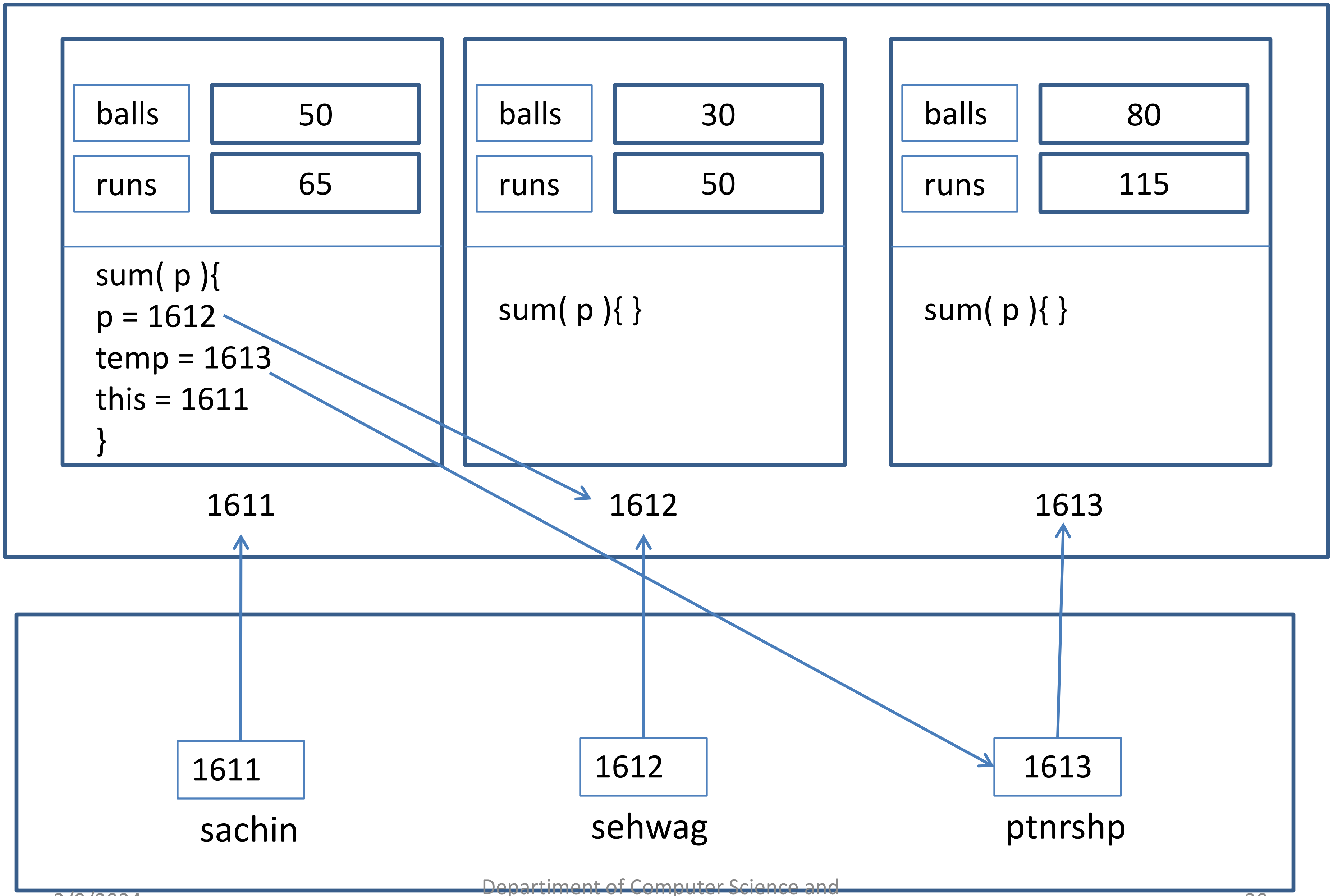
1611
sachin

1612
sehwag

null
ptnrshp

Main block

Memory



balls 50
runs 65

sum(p){
p = 1612
temp = 1613
this = 1611
}

1611

balls 30
runs 50

sum(p){ }

1612

balls 80
runs 115

sum(p){ }

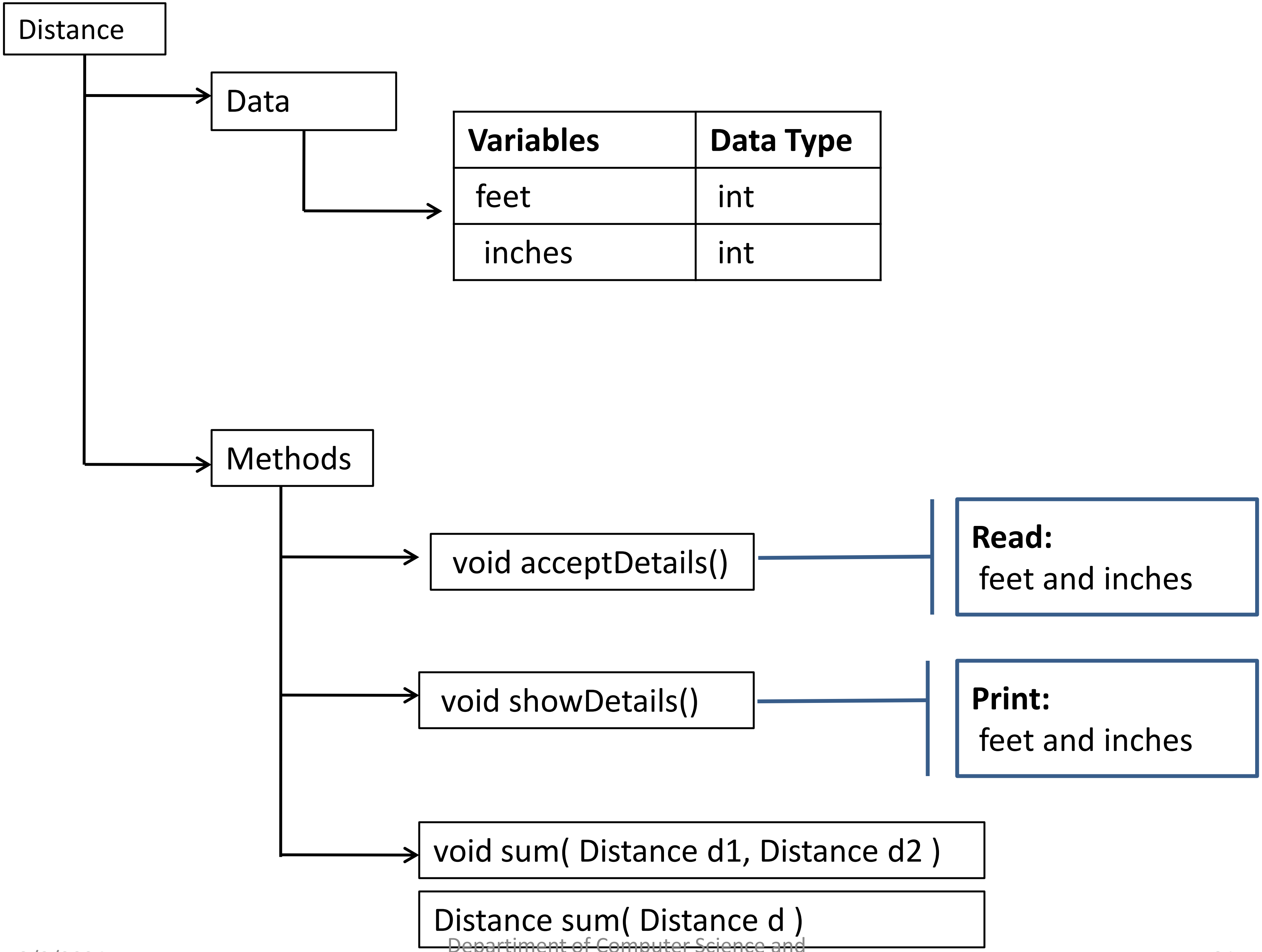
1613

1611
sachin

1612
sehwag

1613
ptnrshp

Main block



Distance

Data

Variables	Data Type
feet	int
inches	int

Methods

void acceptDetails()

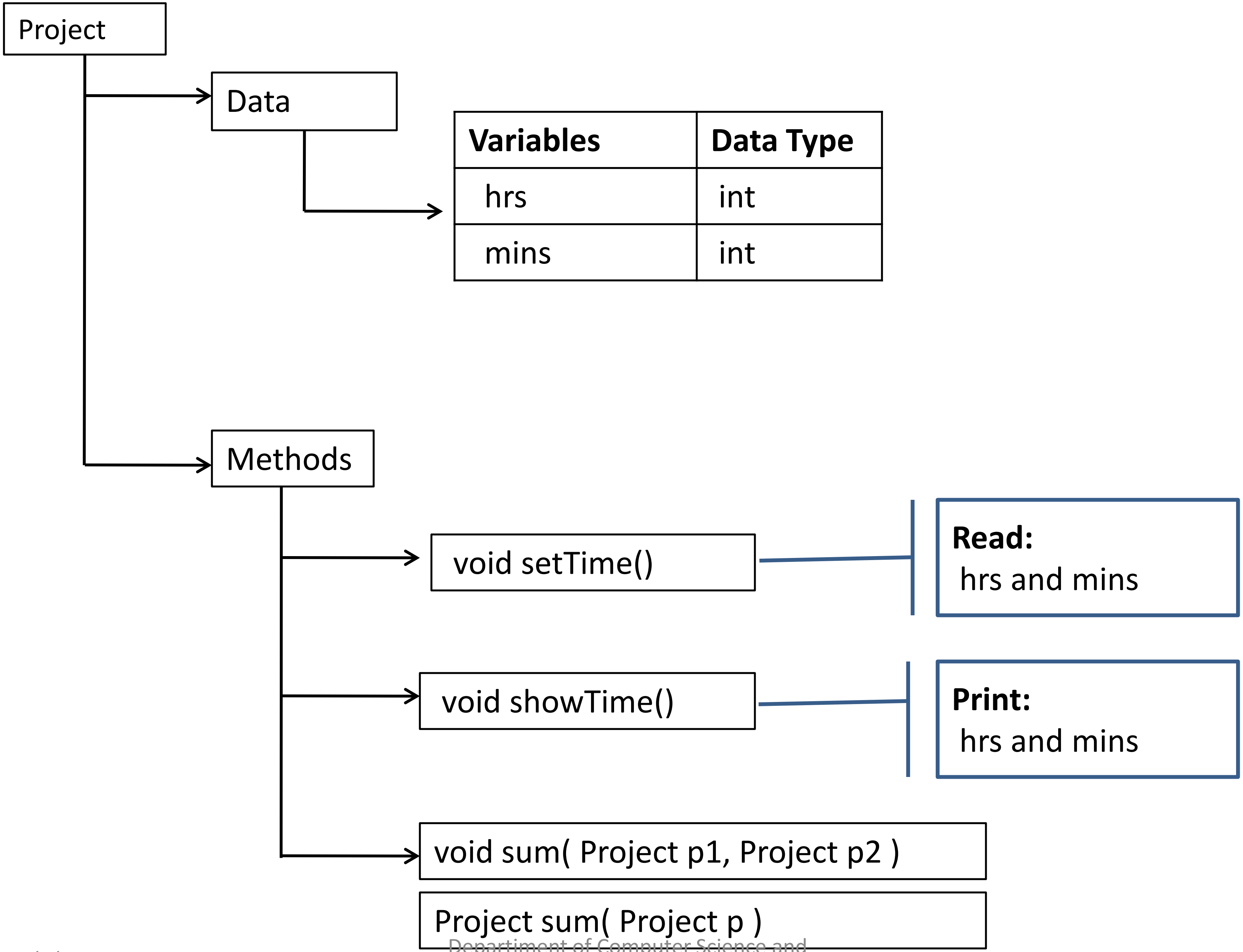
Read:
feet and inches

void showDetails()

Print:
feet and inches

void sum(Distance d1, Distance d2)

Distance sum(Distance d)



Project

Data

Variables	Data Type
hrs	int
mins	int

Methods

void setTime()

Read:
hrs and mins

void showTime()

Print:
hrs and mins

void sum(Project p1, Project p2)

Project sum(Project p)

Static Methods

There are 3 types of variables:

Type	Description
Local Variables	Declared with in the methods and loops
Instance variables	Declared with in the class and each object contains a own copy of instance variables
Static / Class variables	Declared with static keyword and it becomes the class variable, not associate with any object. To access static variables we need static methods

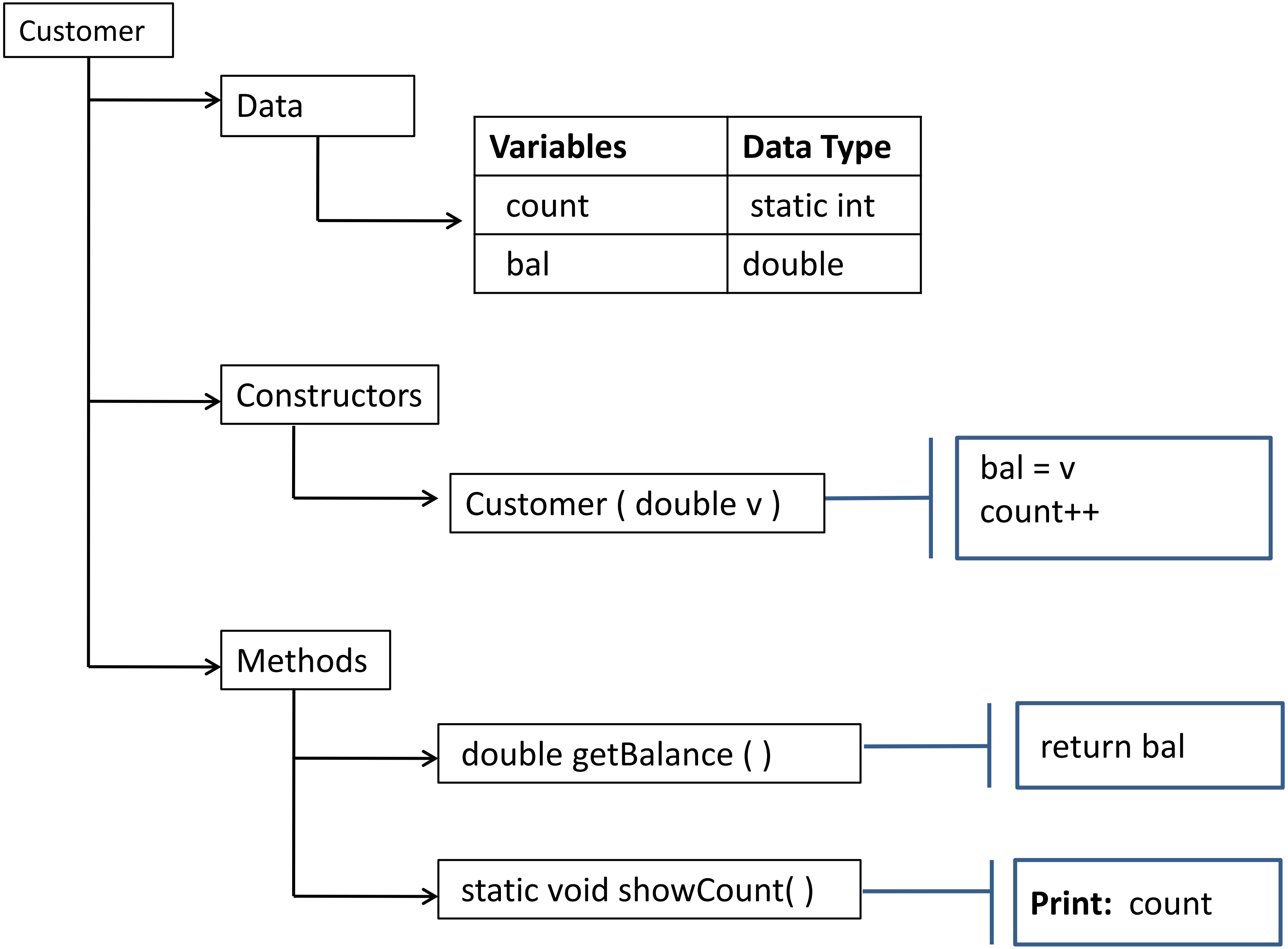
There can be:

✓ static block

✓ static variables

✓ static methods

✓ static class



Customer

Data

Variables	Data Type
count	static int
bal	double

Constructors

Customer (double v)

bal = v
count++

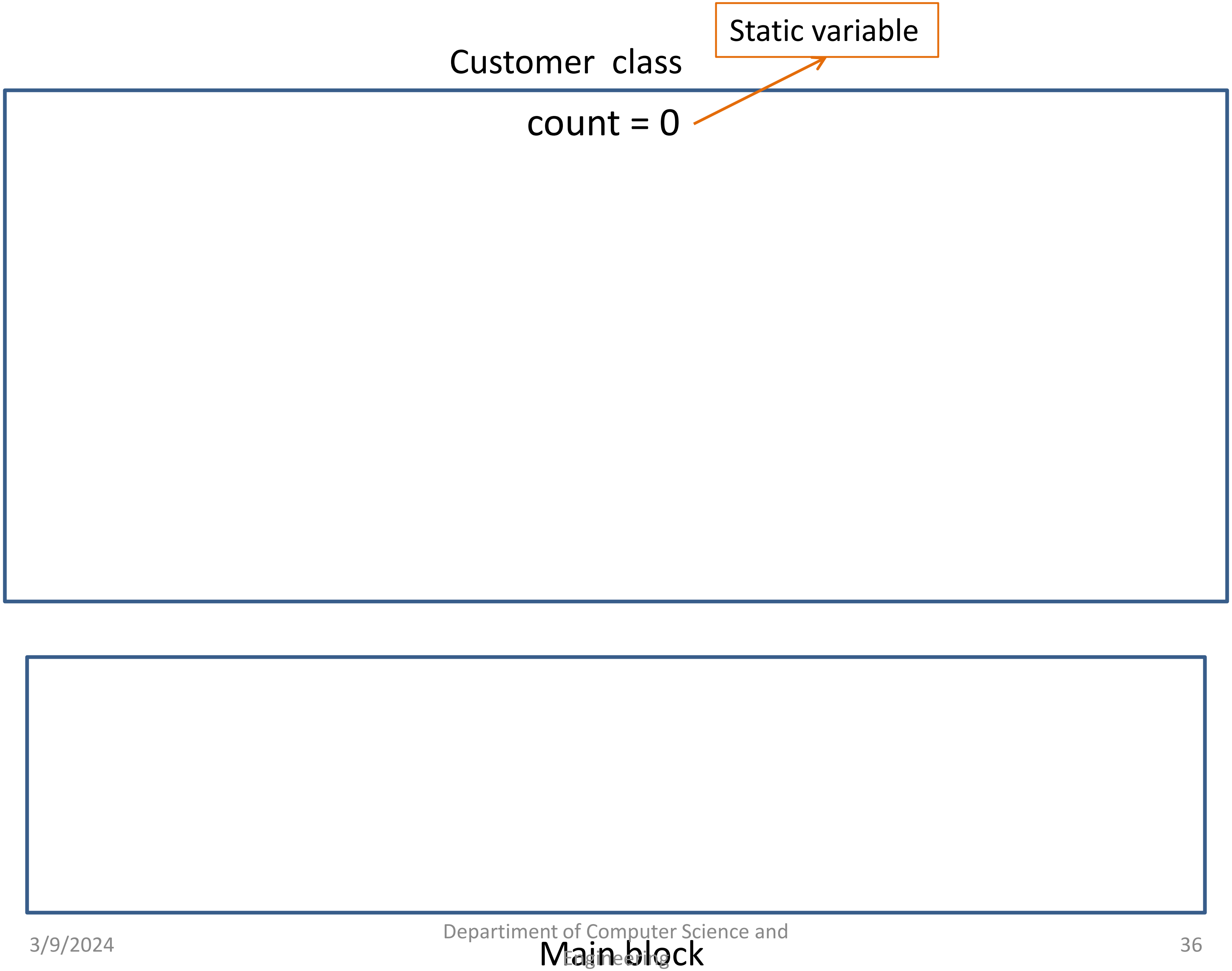
Methods

double getBalance ( )

return bal

static void showCount( )

Print: count

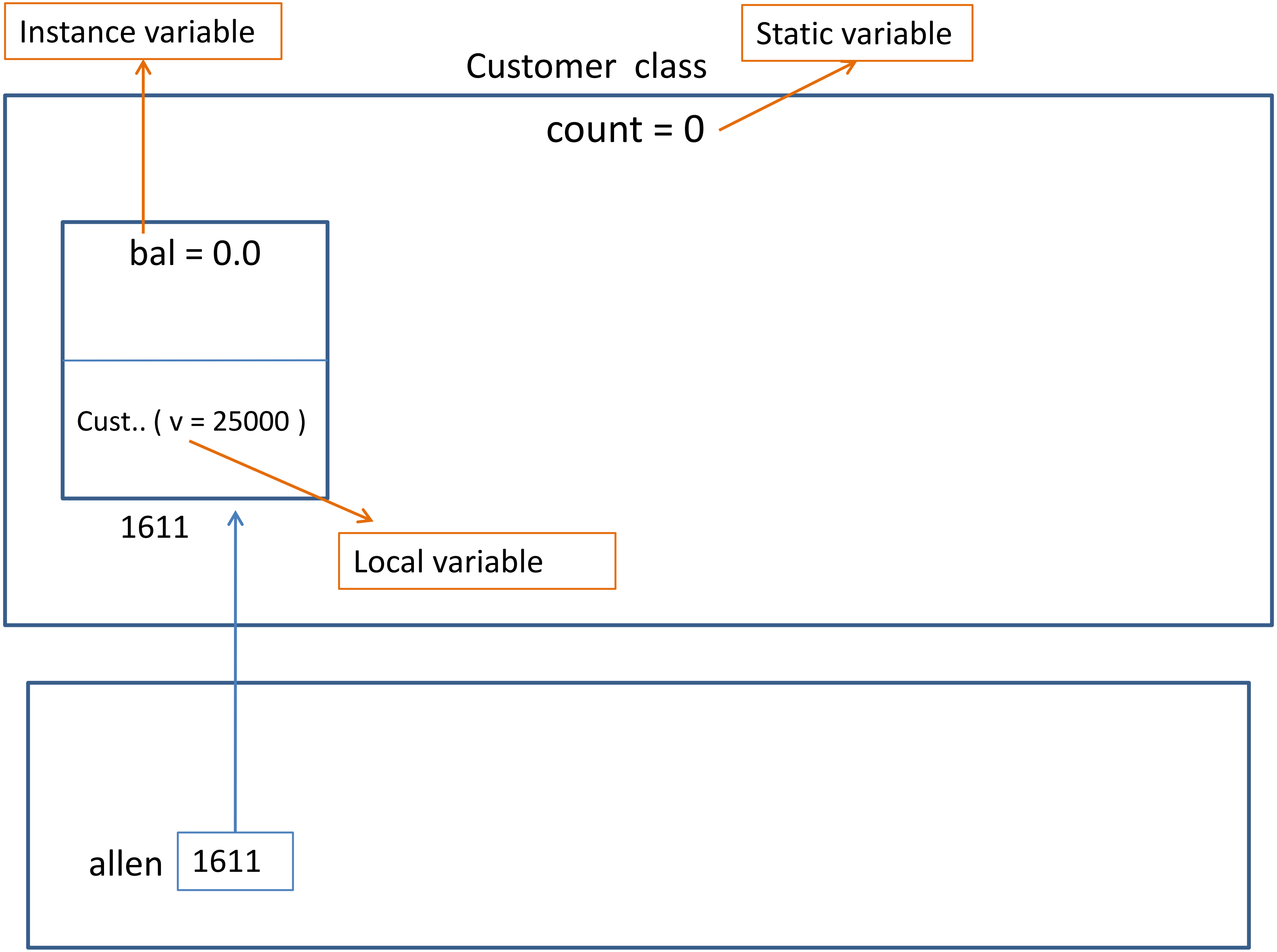


Customer class

count = 0

Static variable

Main block



Instance variable

Customer class

Static variable

count = 0

bal = 0.0

Cust.. (v = 25000)

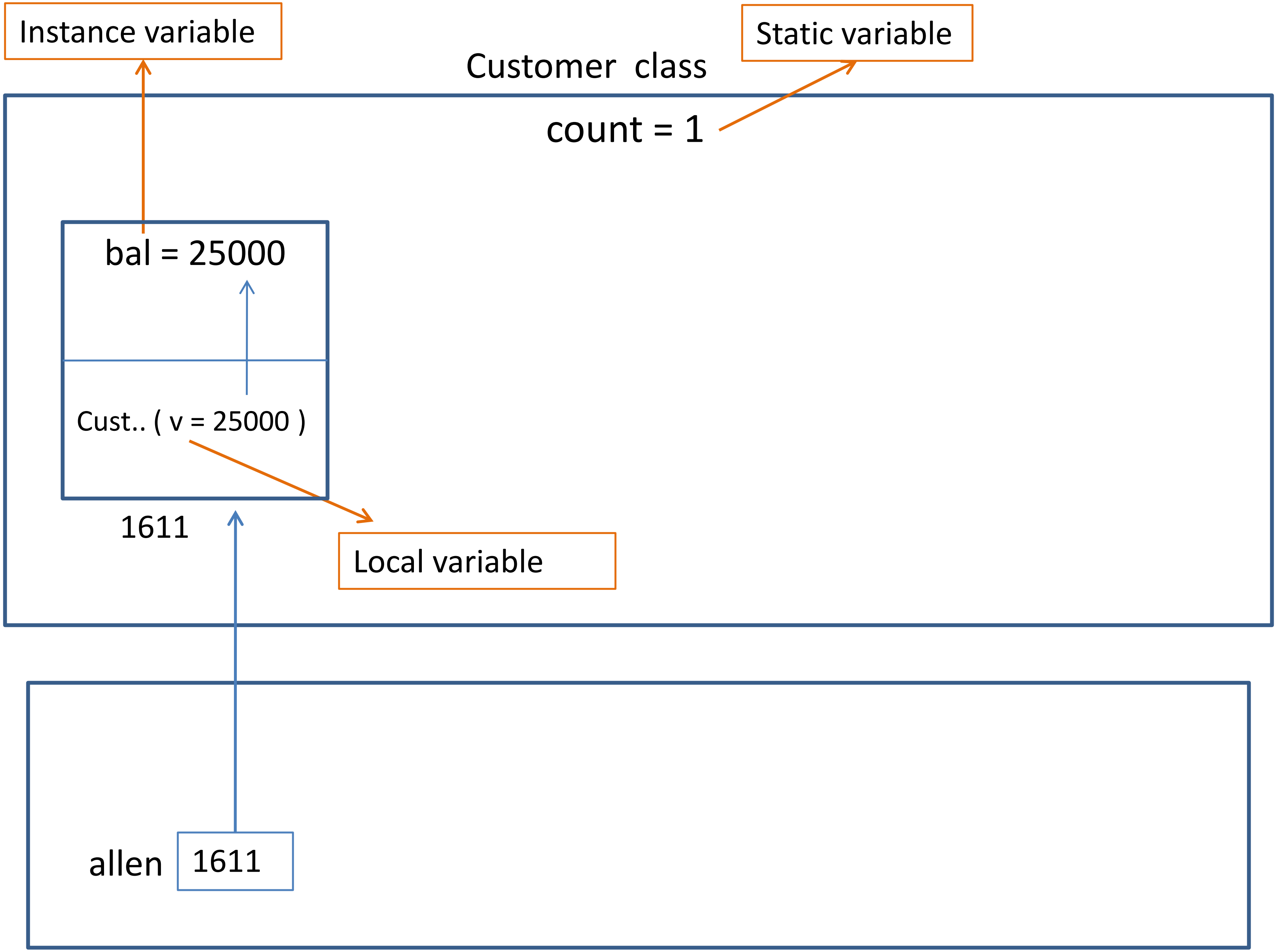
1611

Local variable

allen

1611

Main block



Instance variable

Customer class

Static variable

count = 1

bal = 25000

Cust.. (v = 25000)

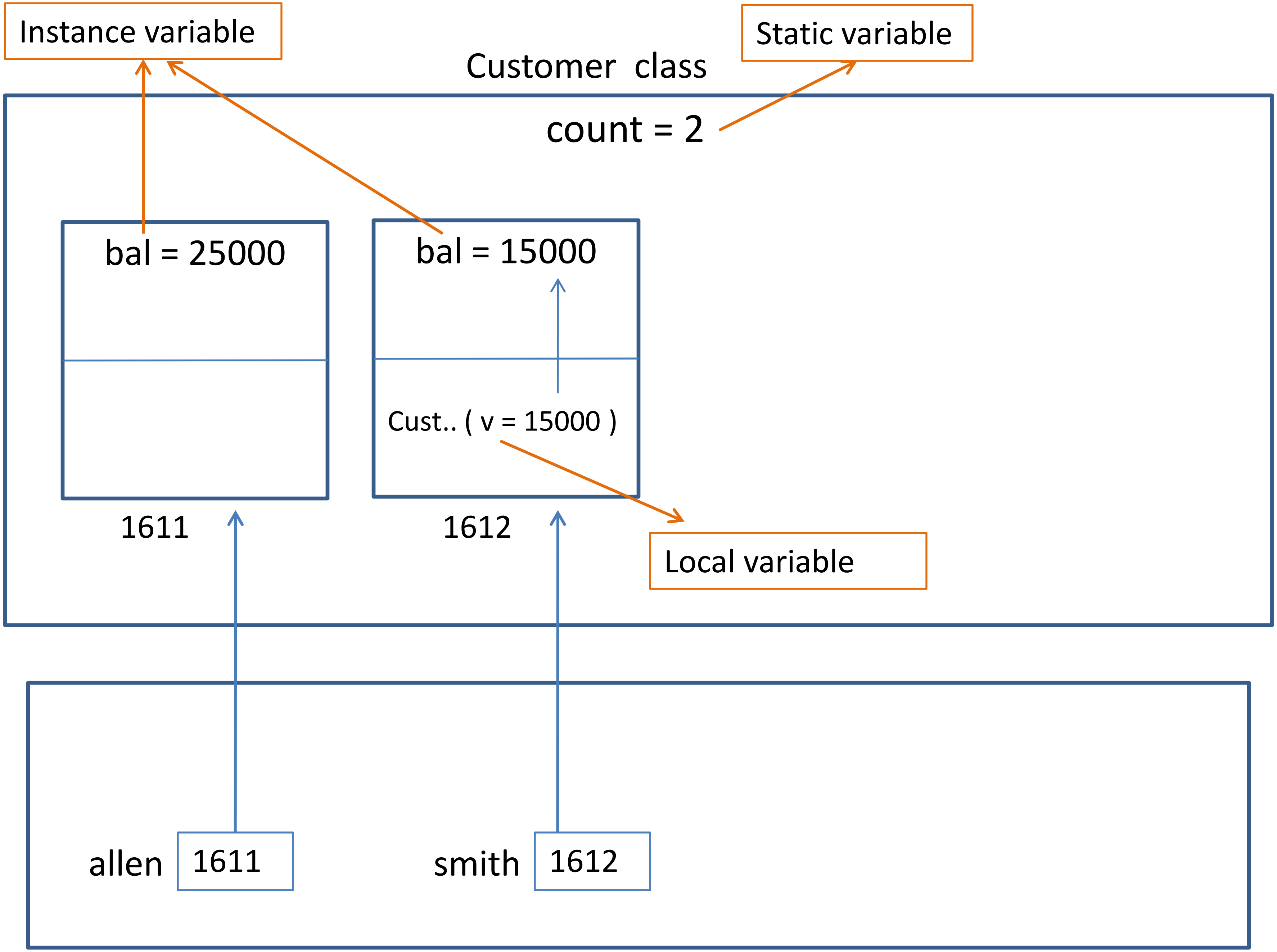
1611

Local variable

allen

1611

Main block



Instance variable

Customer class

Static variable

count = 2

bal = 25000

bal = 15000

Cust.. (v = 15000)

1611

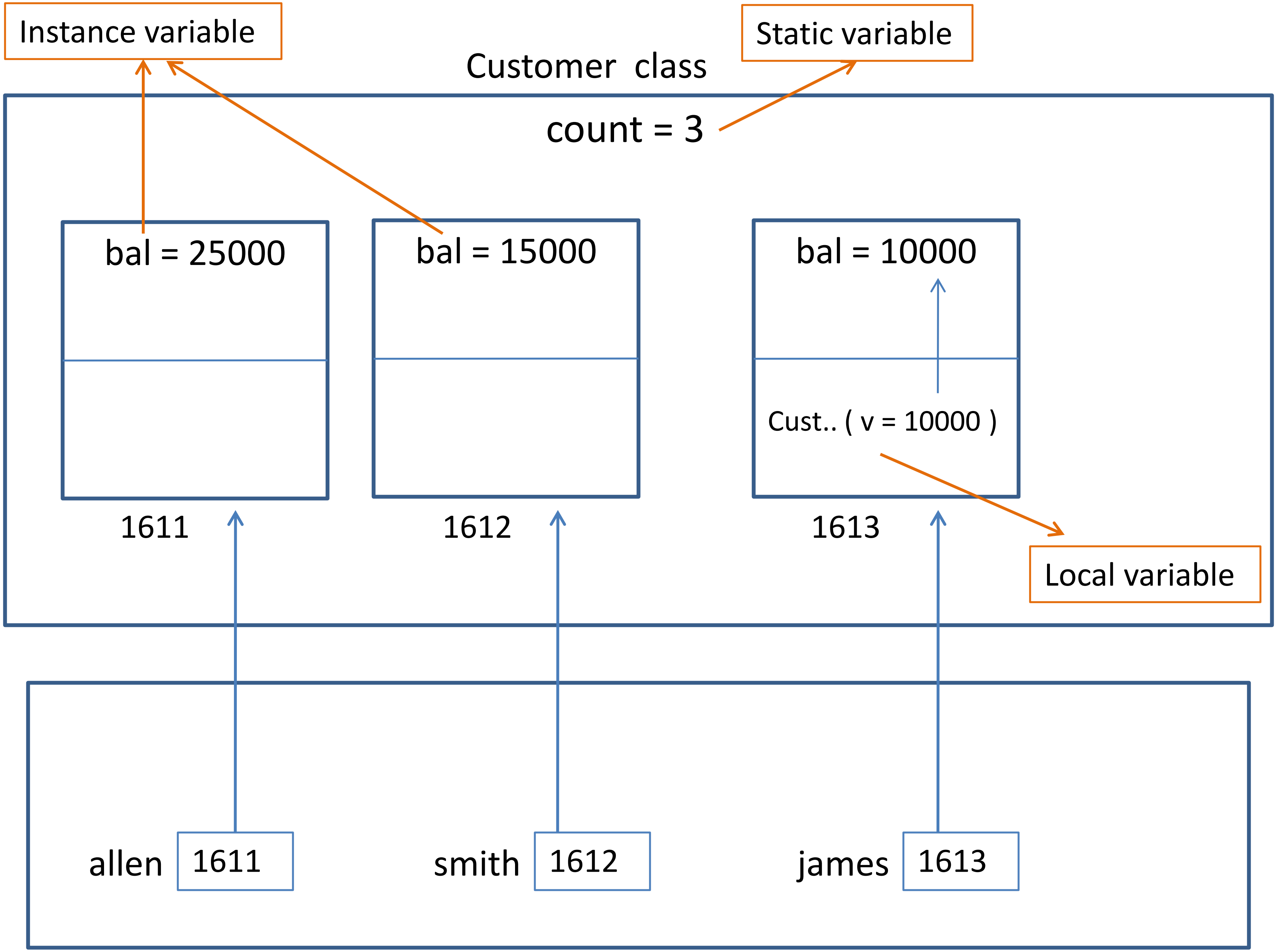
1612

Local variable

allen 1611

smith 1612

Main block


Instance variable
Static variable
Customer class
count = 3
bal = 25000
bal = 15000
bal = 10000
Cust.. (v = 10000)
1611
1612
1613
Local variable
allen
1611
smith
1612
james
1613
Main block

When to use static methods?

- When you have code that can be shared across all instances of the same class, put that portion of code into static method.
- They are basically used to access static field(s) of the class.

Instance method vs Static method

- Instance method can access the instance methods and instance variables directly.
- Instance method can access static variables and static methods directly.
- Static methods can access the static variables and static methods directly.
- Static methods can’t access instance methods and instance variables directly. They must use reference to object. And static method can’t use this keyword as there is no instance for ‘this’ to refer to.

Rules for methods declared as static

- They can only directly call other static methods of their class.
- They can only directly access static variables of their class.
- They can not refer this or super in any way.
- Outside the class static variables or methods can be access by using classname and dot operator.

Using Nested / Inner Class

If you declared a class with in the another class then it is Nested / Inner class

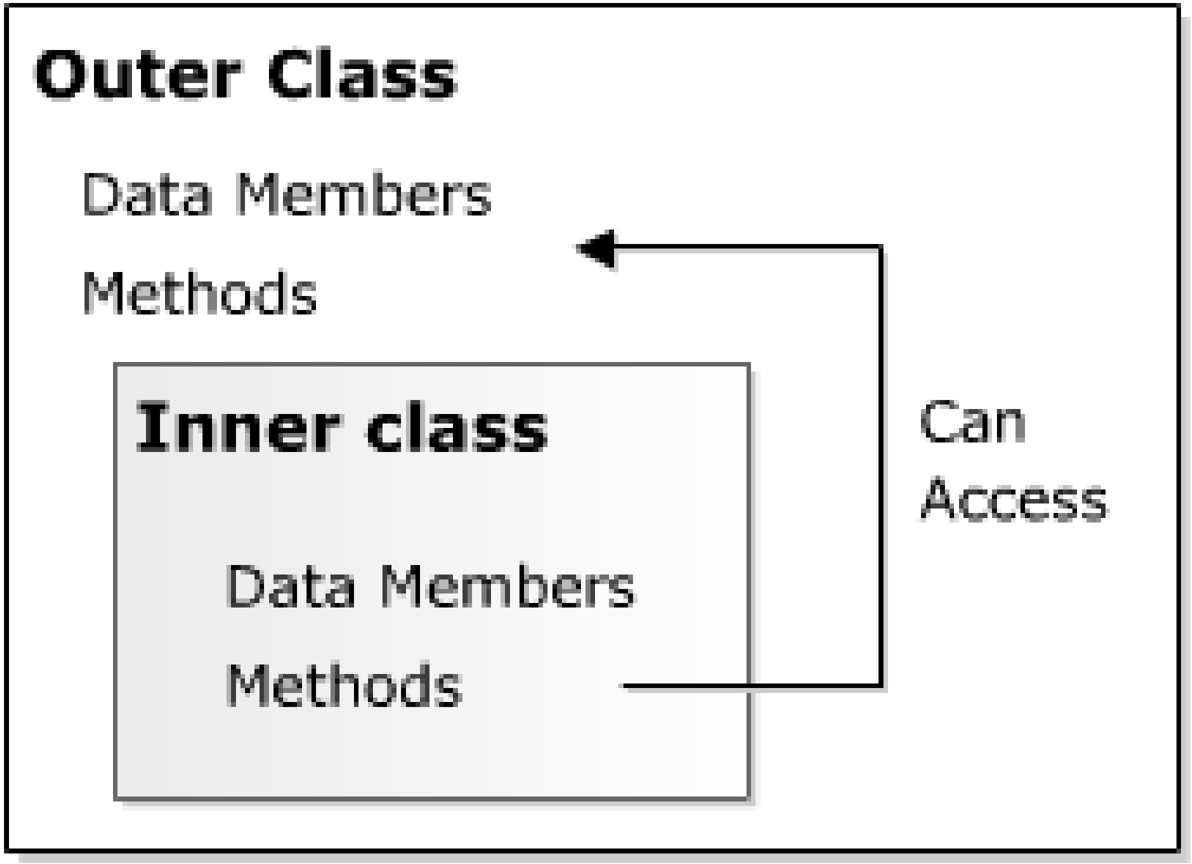
- Nested class can be static
- Nested class can be non-static i.e., Inner Class

Inner class can access the outer class members directly including private members.

For Nested class to access the outer class members, you need to create an Outer class object.

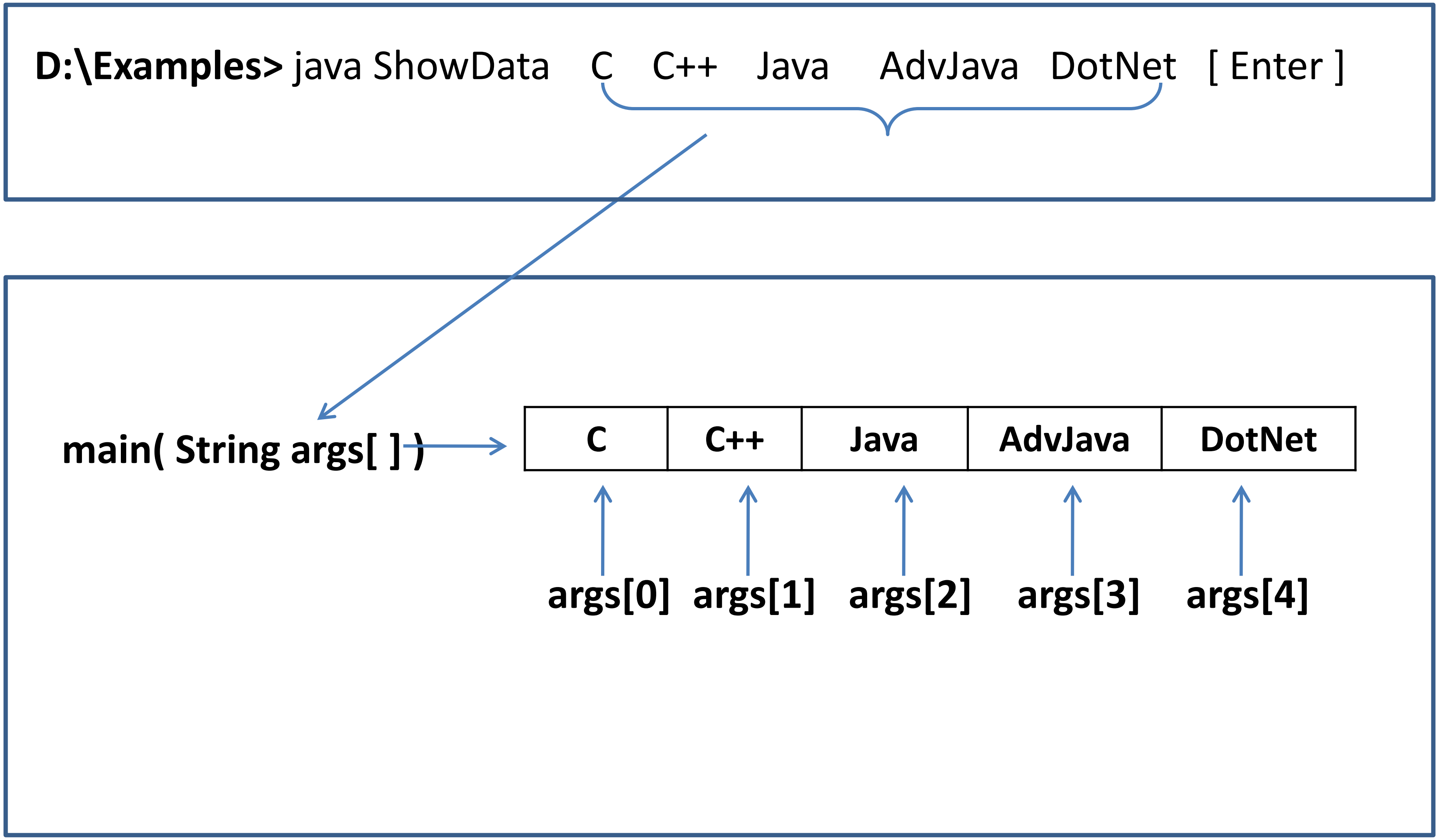
Can Class be declared static?

Generally, A Class cannot be declared static. However, if it is a Nested class then it can be declared static.


Outer Class
Data Members
Methods
Inner class
Data Members
Methods
Can
Access

Command-line arguments in main method

```
D:\Examples> java ShowData    C    C++    Java     AdvJava   DotNet
```



```
D:\Examples> java ShowData   C   C++   Java   AdvJava   DotNet   [ Enter ]
```

main(String args[ ])

C	C++	Java	AdvJava	DotNet
args[0]	args[1]	args[2]	args[3]	args[4]

Methods Using Variable Arguments
(varargs)

Methods using variable arguments permit multiple number of arguments in methods.

for example

```
class Statistics {
        public static double average(int... nums) {
                int sum = 0;
                for ( int  x : nums ) {
                        sum += x;
                }
                return ( sum / (double) nums.length);
        }
}
```

Note

- The three dots … (also called ellipses) is the syntax of **vararg.**
- The vararg parameter is treated as an array.
- The vararg must be the last argument in the method.

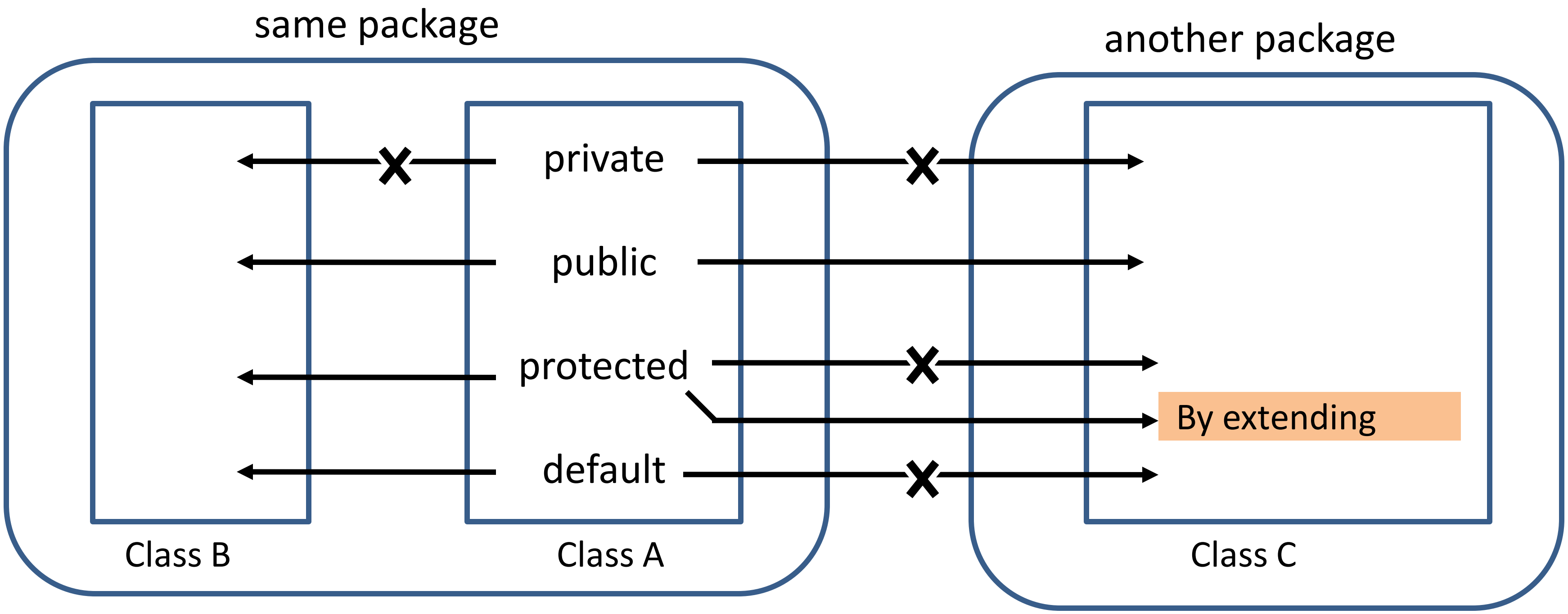
Usage

```
double averageGrade = Statistics.average(4, 3, 4);
double averageAge = Statistics.average(24, 32, 27, 18);
```

Access Specifiers in Java:

- There are four access specifiers in java :
  private, public, protected and default.

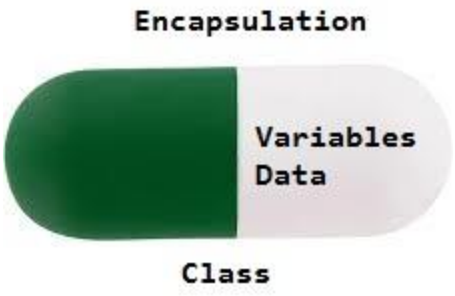
- Let’s see how these Access specifiers protects the data



	Private	Default	Protected	Public
Same class	✓	✓	✓	✓
Same package Sub – class	☒	✓	✓	✓
Same package Non – subclass	☒	✓	✓	✓
Different package Sub – class	☒	☒	✓	✓
Different package Non – subclass	☒	☒	☒	✓

Encapsulation

- Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class in Java.
- Java Encapsulation is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.



Example of Encapsulation

```
// Java Program to demonstrate
// Java Encapsulation

// Person Class
class Person {
        // Encapsulating the name and age
        // only approachable and used using
        // methods defined
        private String name;
        private int age;

        public String getName() { return name; }

        public void setName(String name) { this.name =
name; }

        public int getAge() { return age; }

        public void setAge(int age) { this.age = age; }
}
```

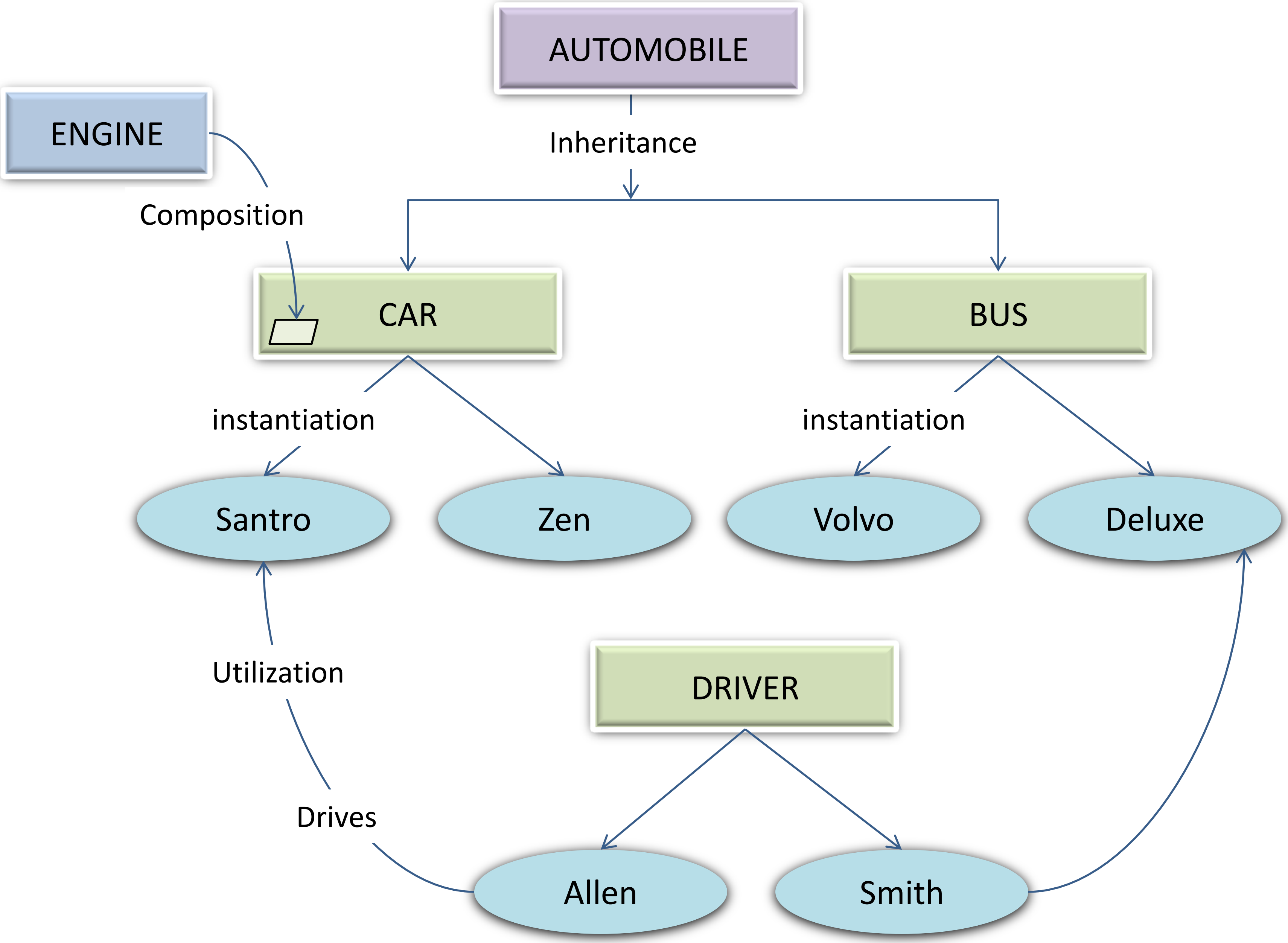
```
// Driver Class
public class Main {
        // main function
        public static void main(String[] args)
        {
                // person object created
                Person person = new
Person();
                person.setName("John");
                person.setAge(30);

                // Using methods to get
the values from the
                // variables
                System.out.println("Name:
" + person.getName());
                System.out.println("Age: "
+ person.getAge());
        }
}
```

Relationship among objects

Types of Relationships

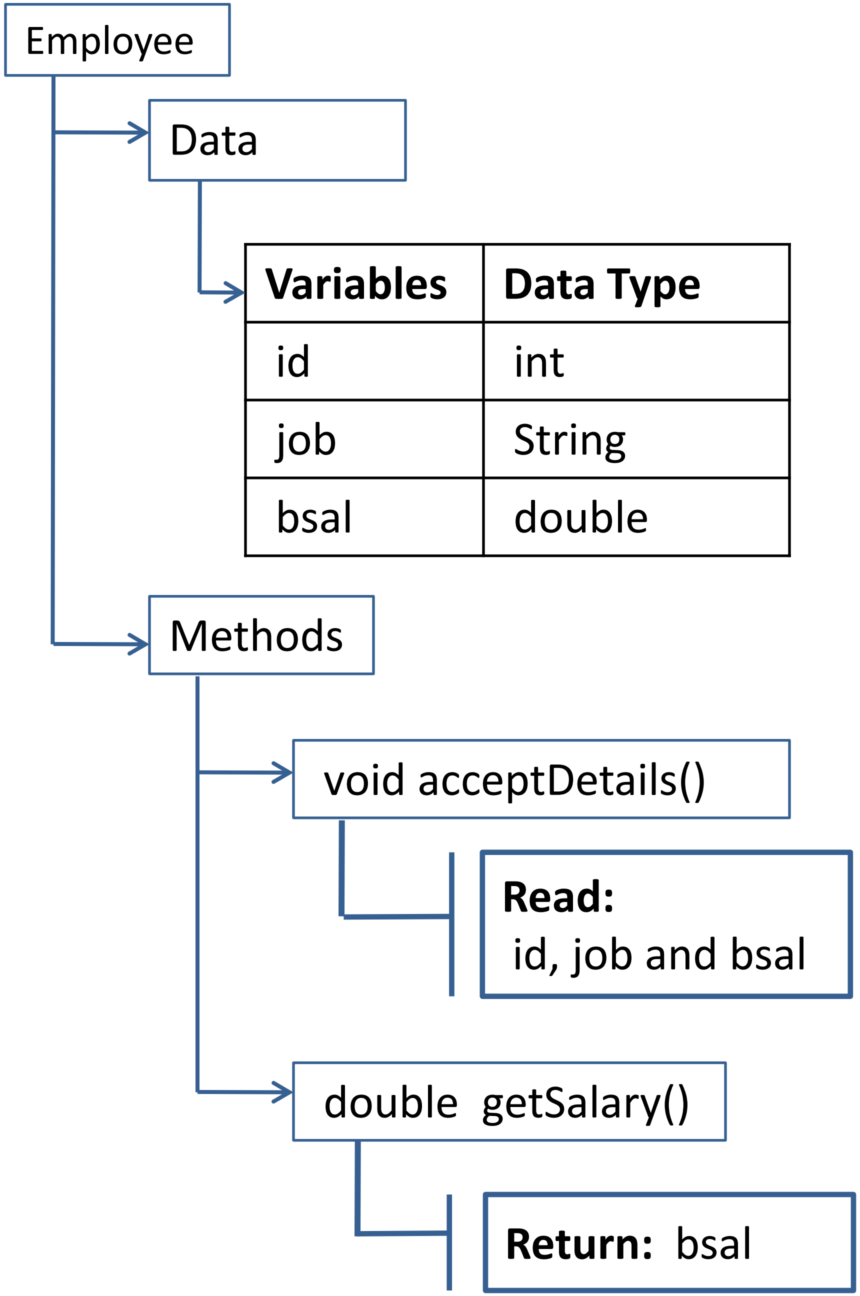
- ❑ Instantiation
- ❑ Utilization
- ❑ Composition
- ❑ Inheritance


AUTOMOBILE
ENGINE
Inheritance
Composition
CAR
BUS
instantiation
instantiation
Santro
Zen
Volvo
Deluxe
Utilization
DRIVER
Drives
Allen
Smith

There are 3 ways to relate objects of different classes with each other:

- ❑ Using Reference (Utilization)
- ❑ Using Inner Class (Composition)
- ❑ Using Inheritance

Using Reference



Employee

Data

Variables	Data Type
id	int
job	String
bsal	double

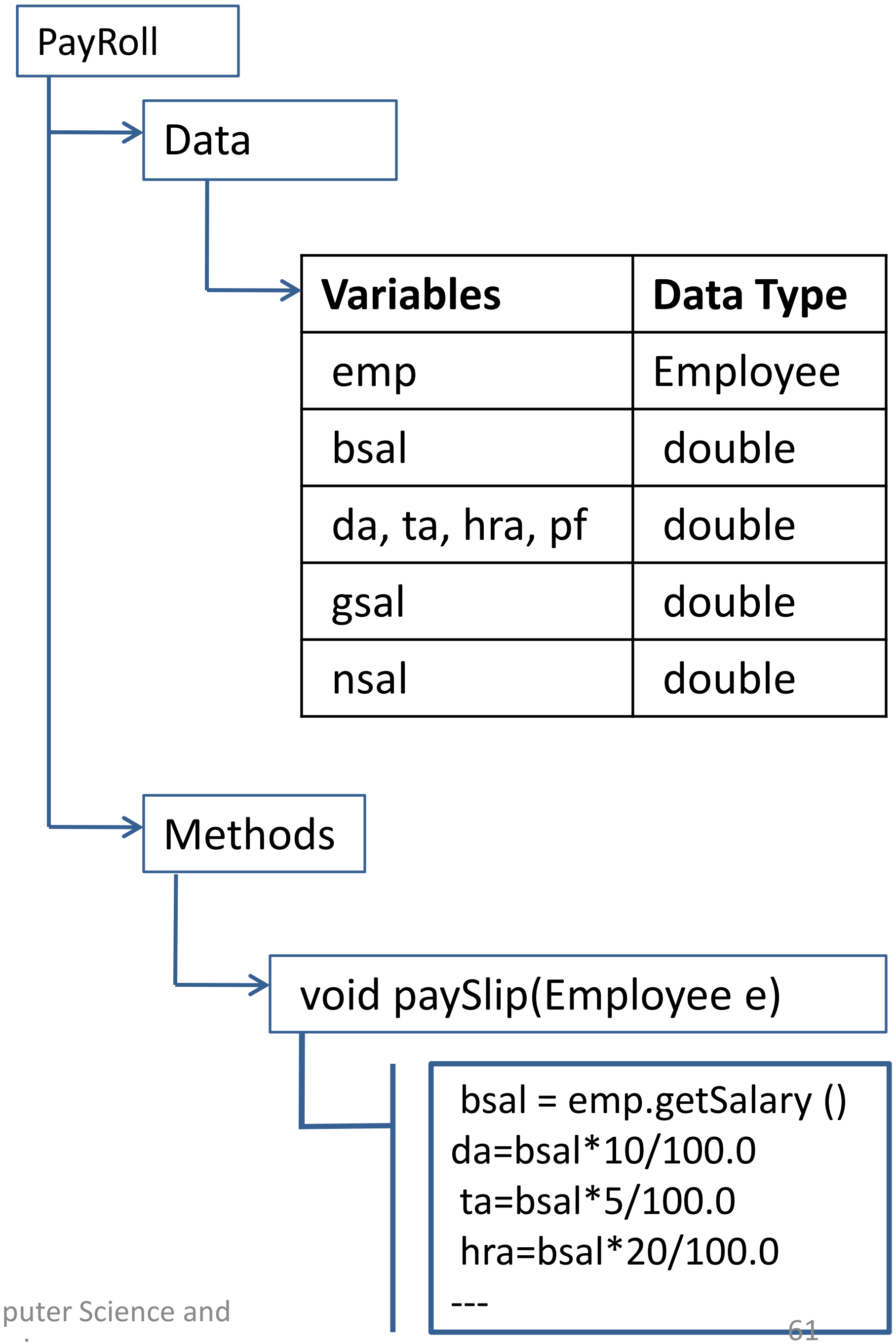
Methods

void acceptDetails()

Read:
id, job and bsal

double getSalary()

Return: bsal



PayRoll

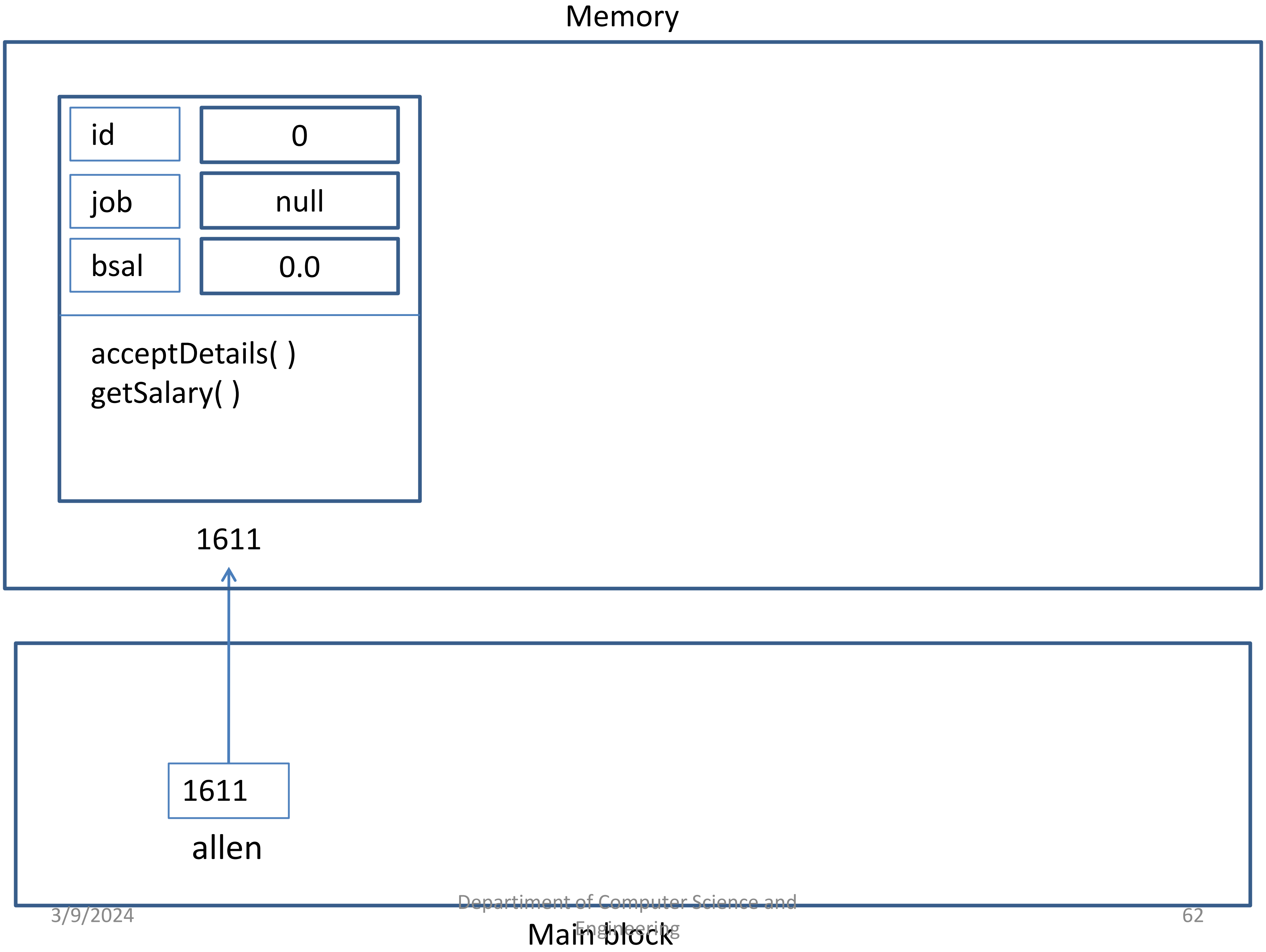
Data

Variables	Data Type
emp	Employee
bsal	double
da, ta, hra, pf	double
gsal	double
nsal	double

Methods

void paySlip(Employee e)

```
bsal = emp.getSalary ()
da=bsal*10/100.0
ta=bsal*5/100.0
hra=bsal*20/100.0
---
```



Memory

id 0

job null

bsal 0.0

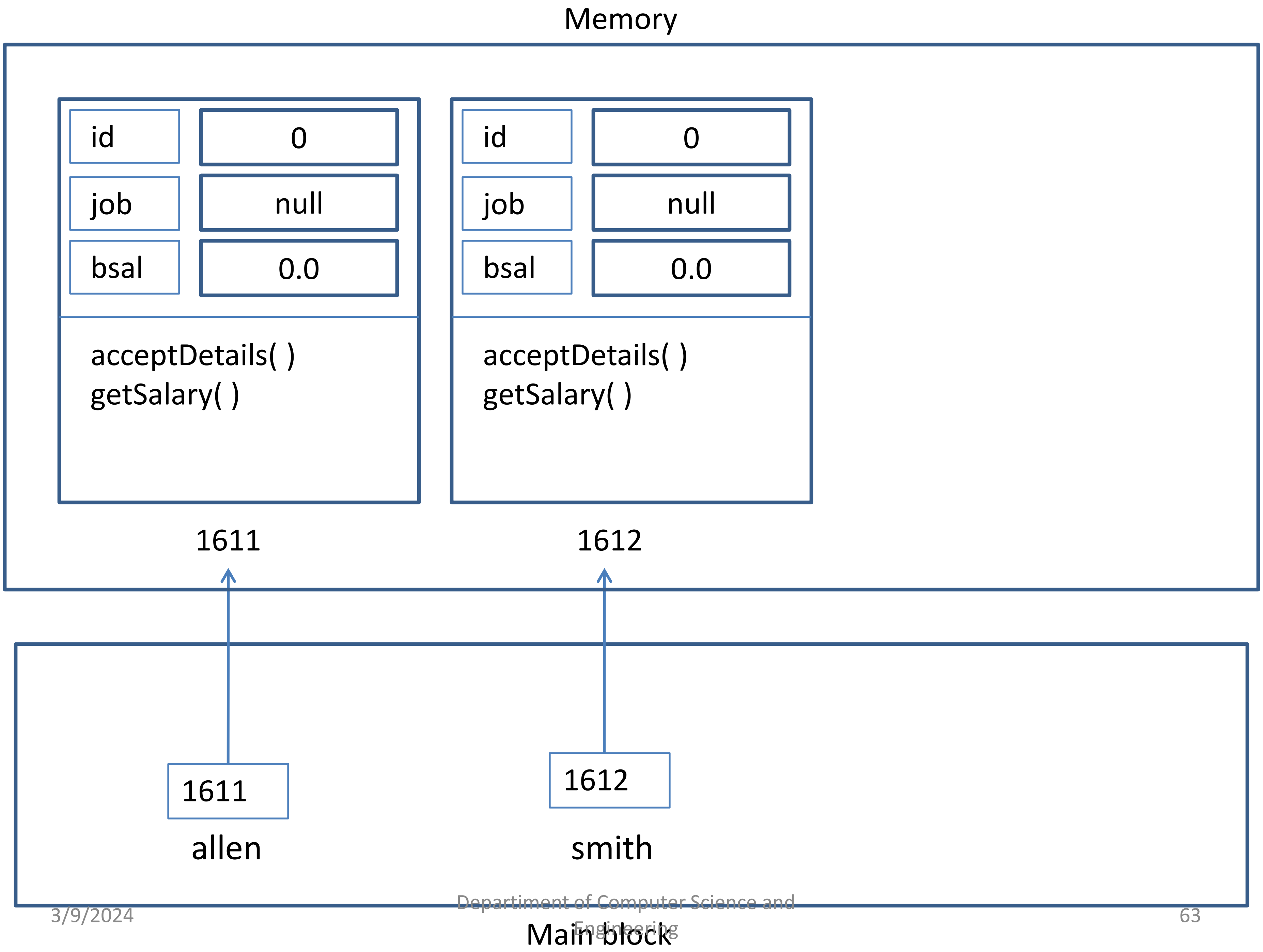
acceptDetails( )
getSalary( )

1611

1611

allen

Main block



Memory

id
0
job
null
bsal
0.0

acceptDetails( )
getSalary( )

1611

id
0
job
null
bsal
0.0

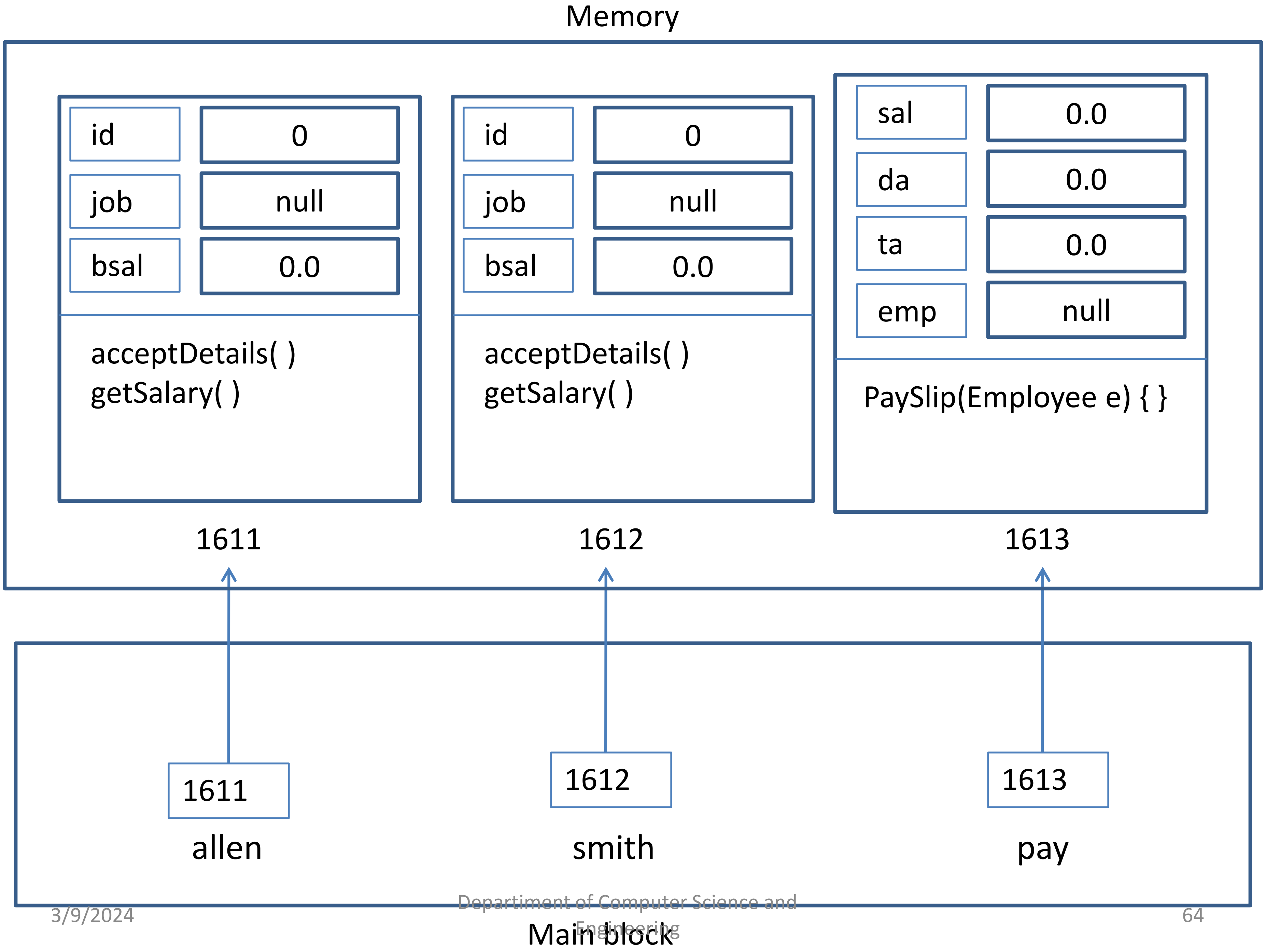
acceptDetails( )
getSalary( )

1612

1611
allen

1612
smith

Main block



Memory

id 0
job null
bsal 0.0

acceptDetails( )
getSalary( )

1611

id 0
job null
bsal 0.0

acceptDetails( )
getSalary( )

1612

sal 0.0
da 0.0
ta 0.0
emp null

PaySlip(Employee e) { }

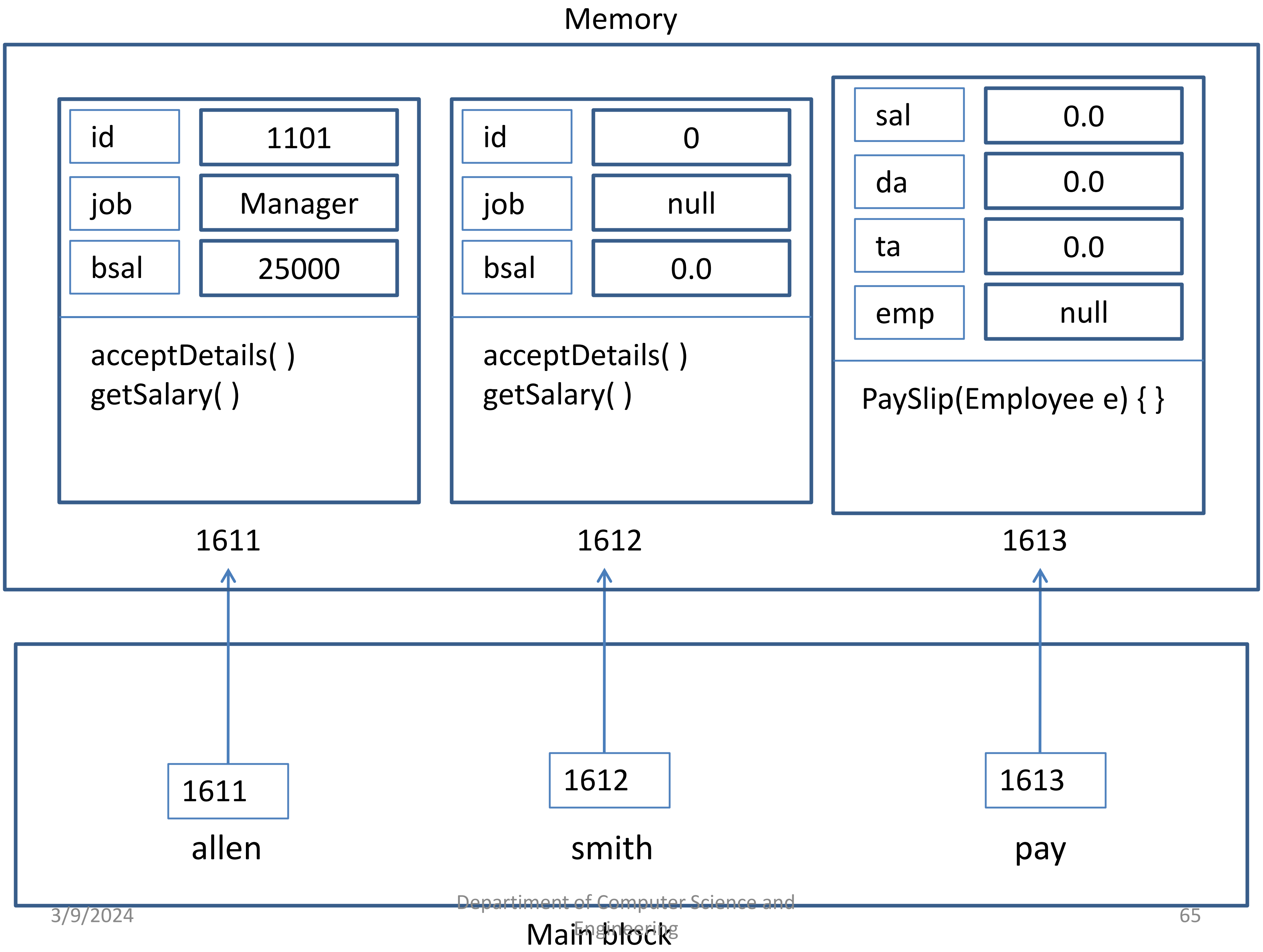
1613

1611
allen

1612
smith

1613
pay

Main block



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	0
job	null
bsal	0.0

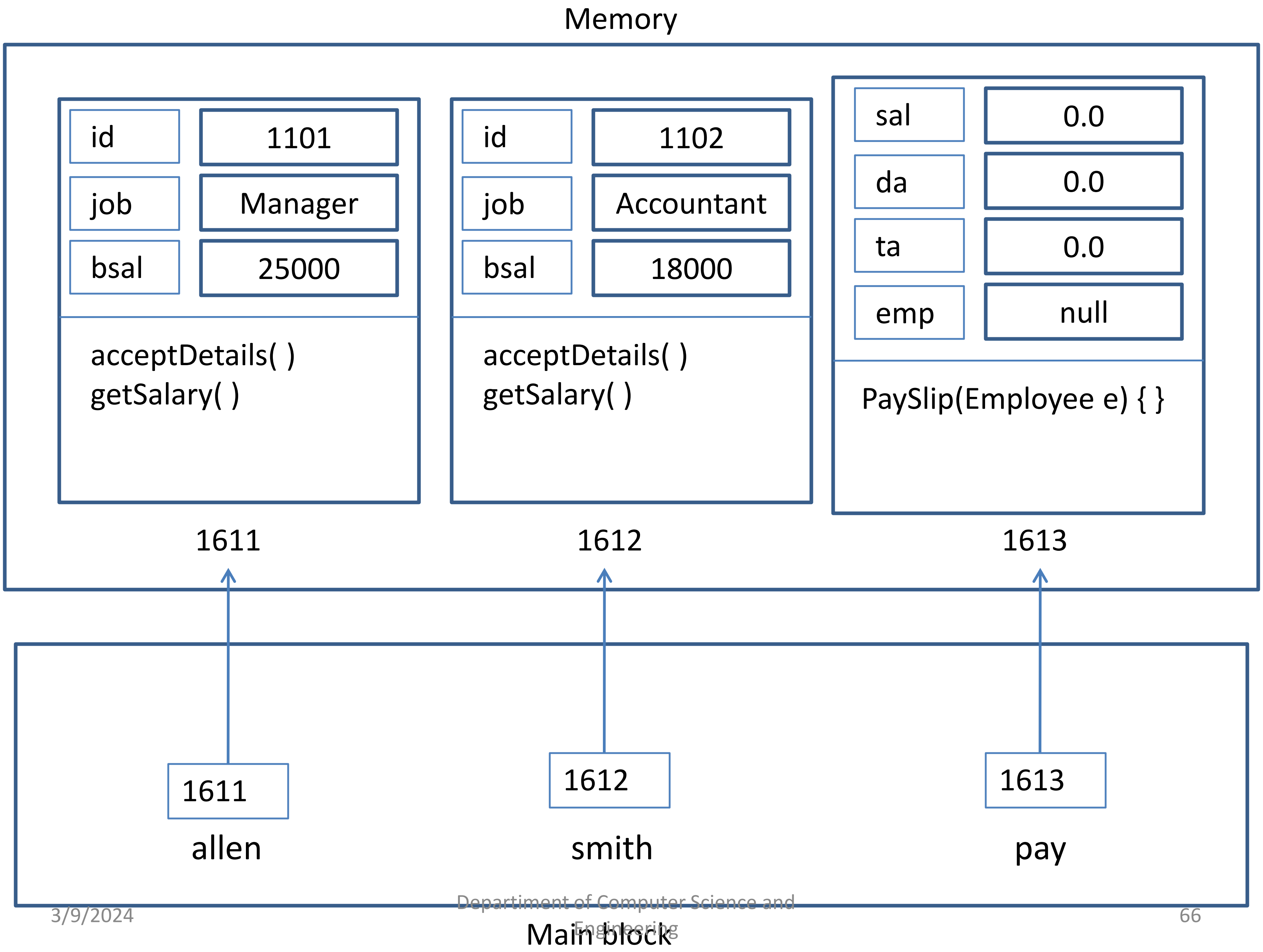
acceptDetails( )
getSalary( )

1612

sal	0.0
da	0.0
ta	0.0
emp	null

PaySlip(Employee e) { }

1613



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

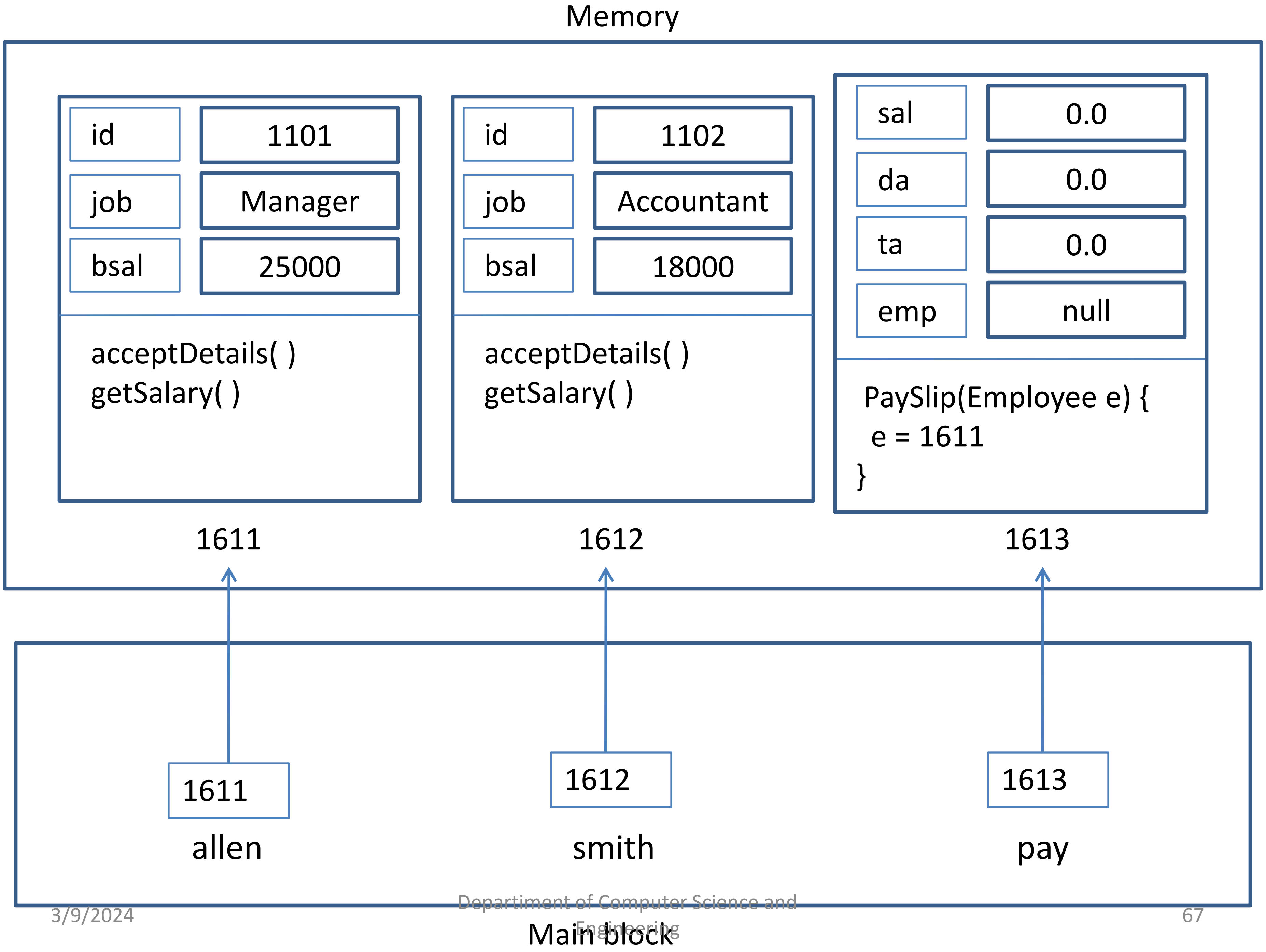
1612

sal	0.0
da	0.0
ta	0.0
emp	null

PaySlip(Employee e) { }

1613

Main block



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

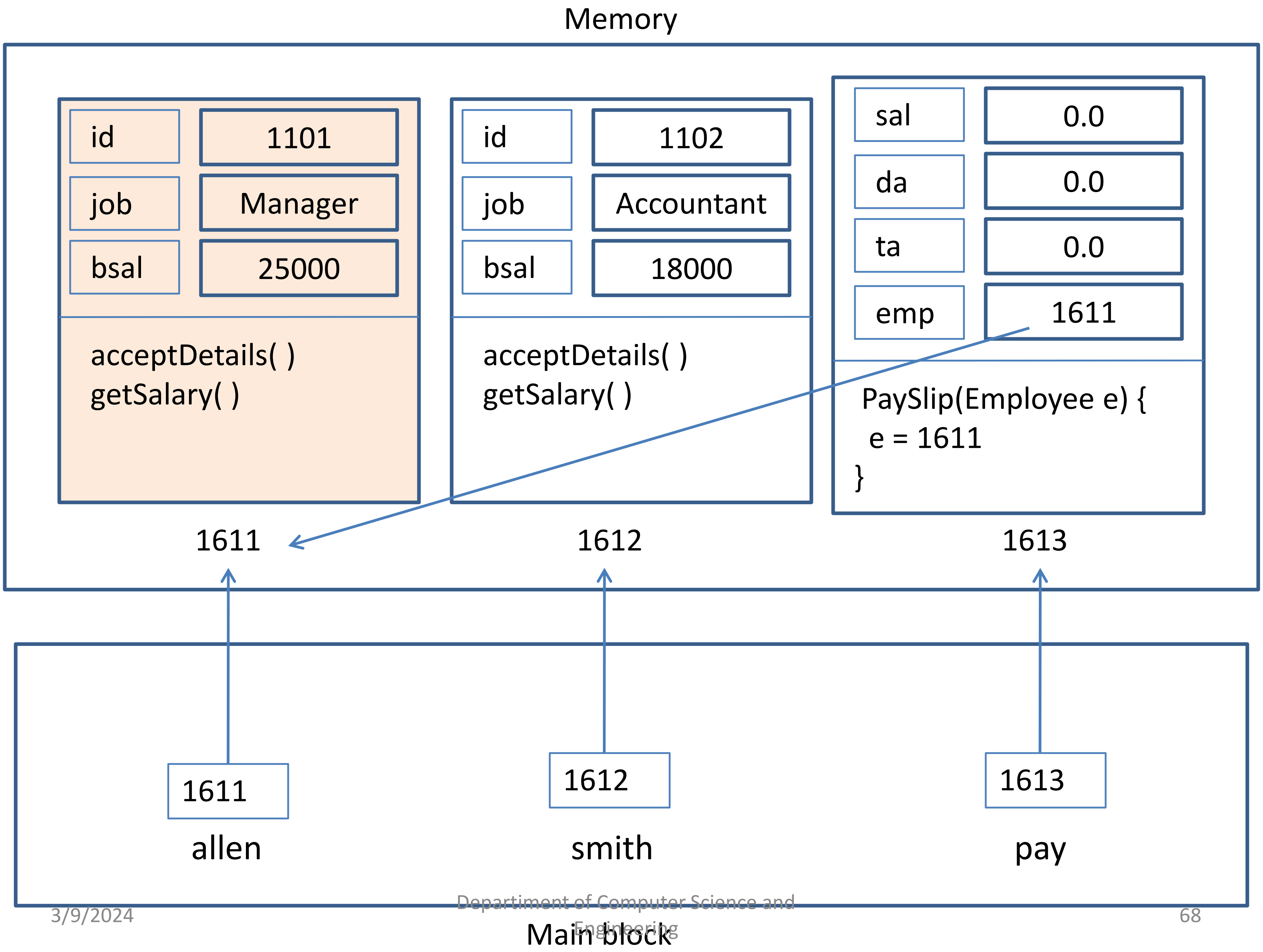
sal	0.0
da	0.0
ta	0.0
emp	null

```
PaySlip(Employee e) {
 e = 1611
}
```

1613

allen	smith	pay
1611	1612	1613

Main block



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

sal	0.0
da	0.0
ta	0.0
emp	1611

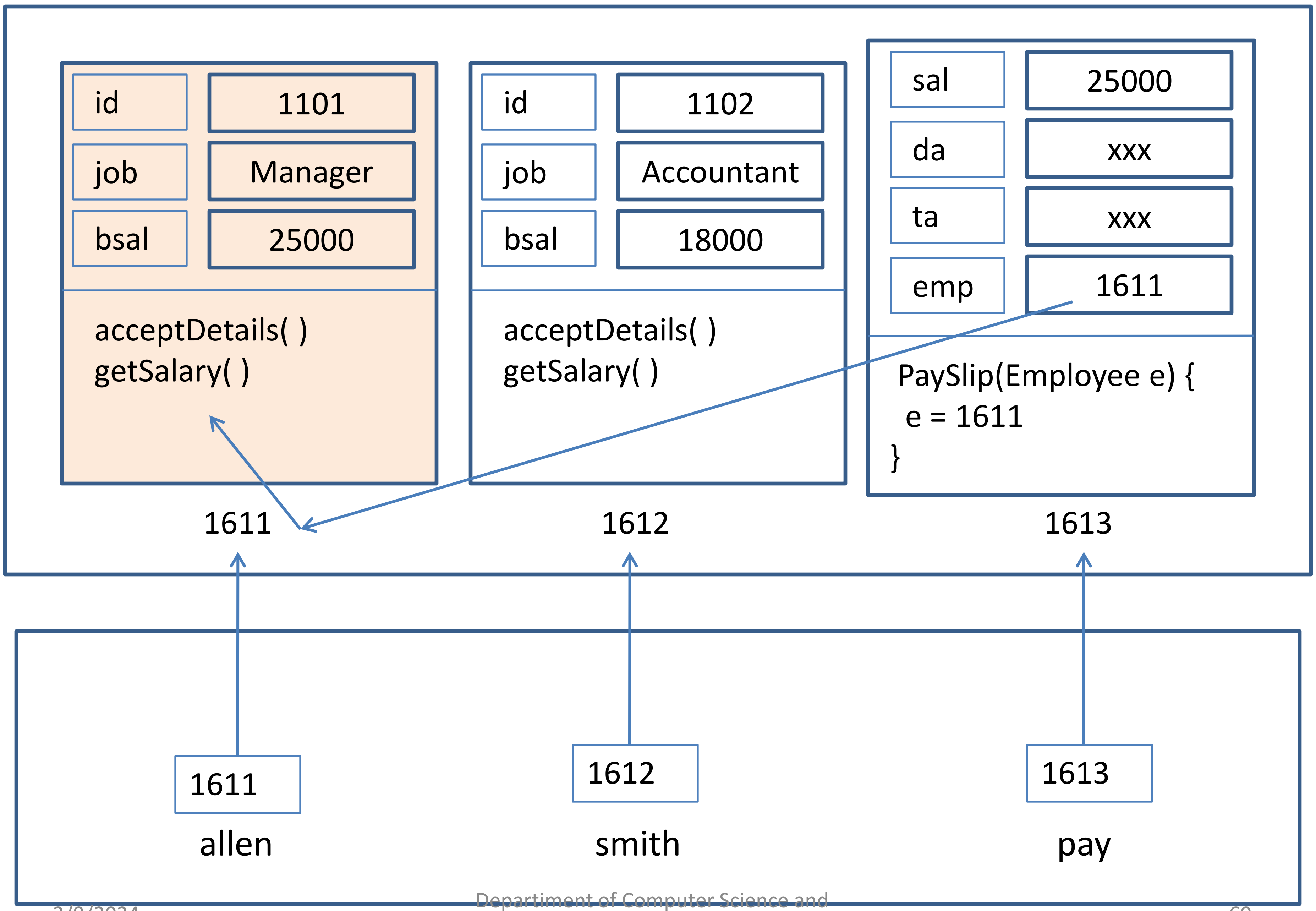
```
PaySlip(Employee e) {
 e = 1611
}
```

1613

1611	1612	1613
allen	smith	pay

Main block

Memory



id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

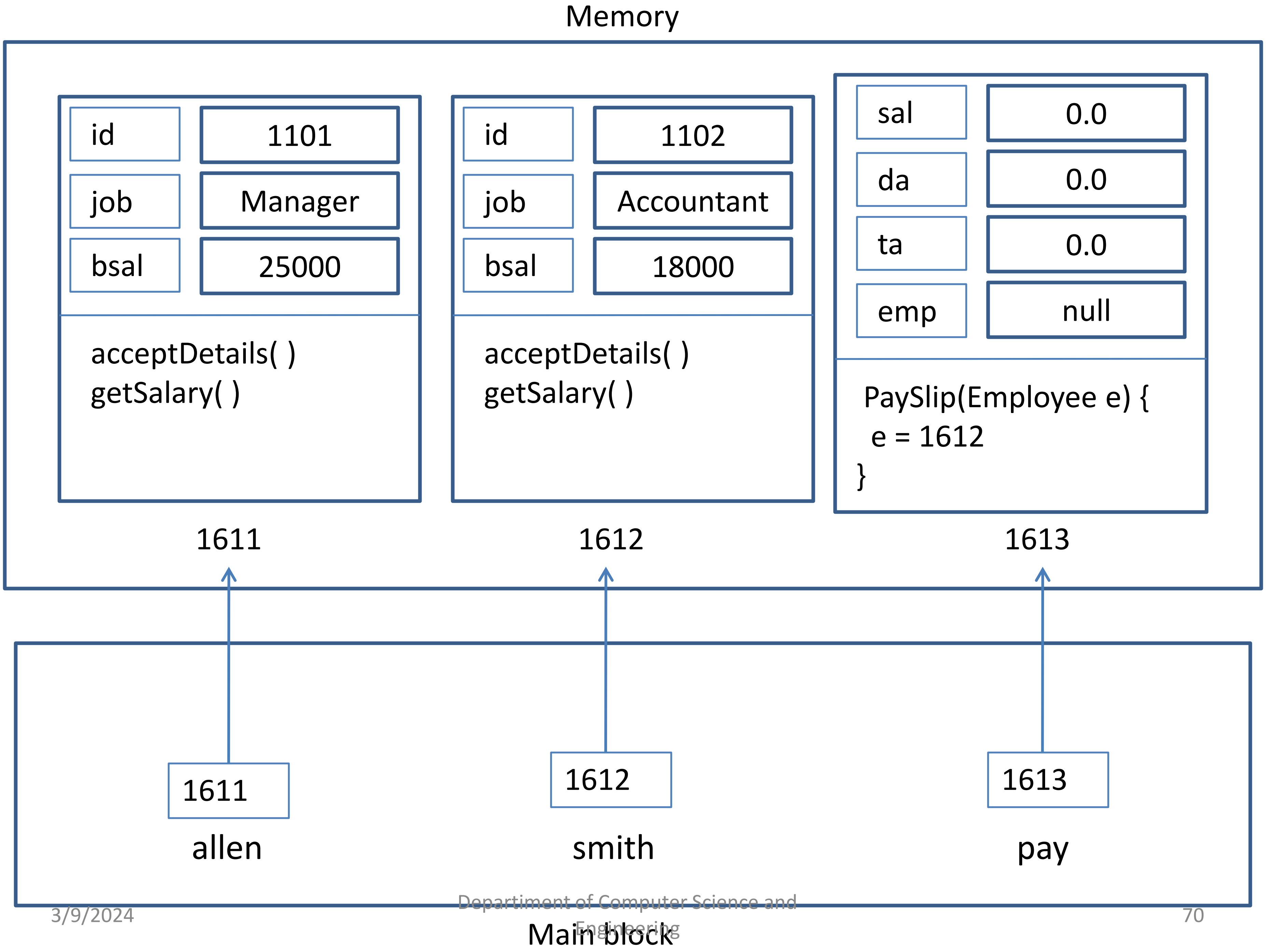
sal	25000
da	xxx
ta	xxx
emp	1611

```
PaySlip(Employee e) {
 e = 1611
}
```

1613

1611	allen
1612	smith
1613	pay

Main block



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

sal	0.0
da	0.0
ta	0.0
emp	null

```
PaySlip(Employee e) {
 e = 1612
}
```

1613

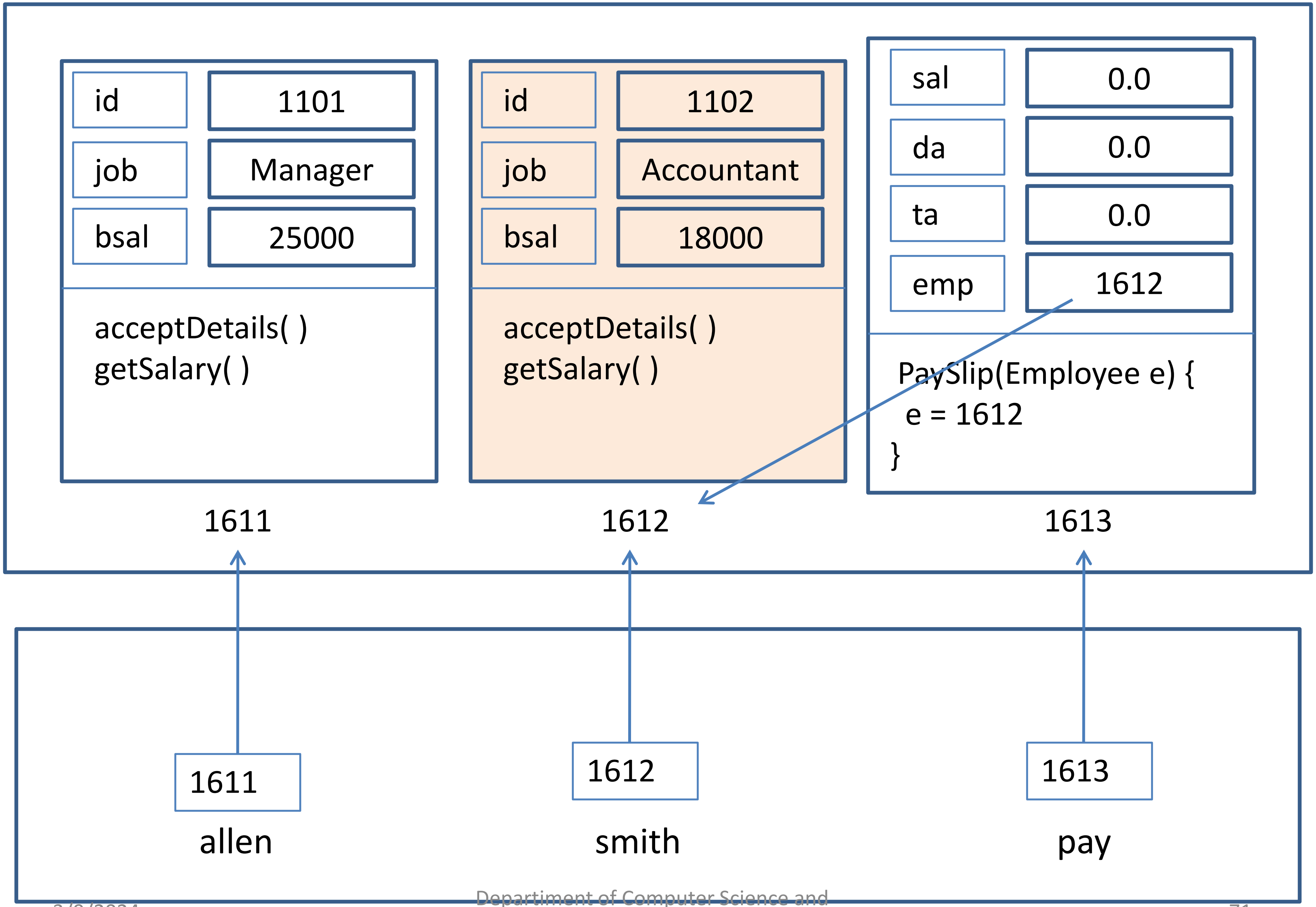
1611
allen

1612
smith

1613
pay

Main block

Memory



id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

sal	0.0
da	0.0
ta	0.0
emp	1612

```
PaySlip(Employee e) {
  e = 1612
}
```

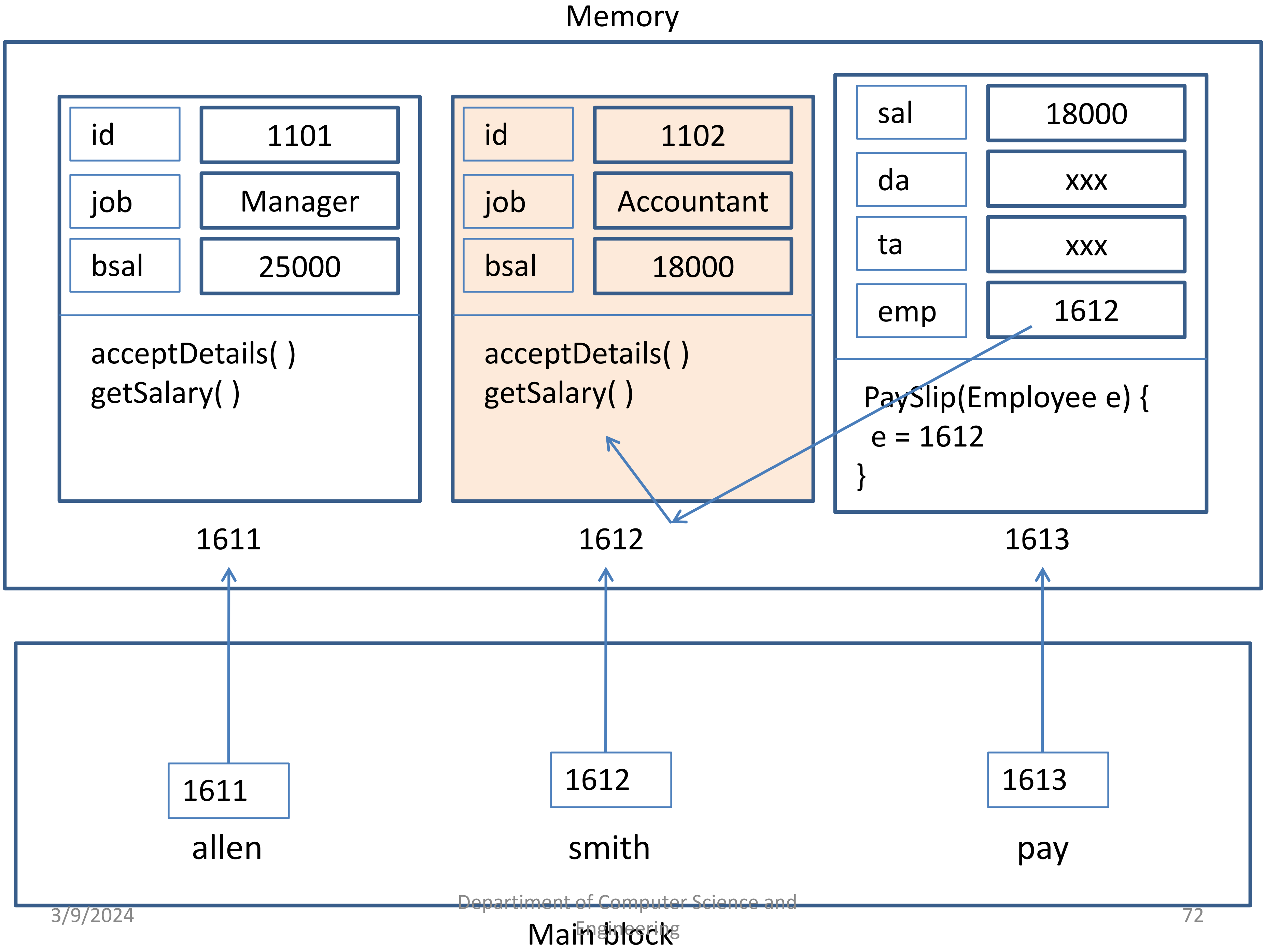
1613

1611
allen

1612
smith

1613
pay

Main block



Memory

id	1101
job	Manager
bsal	25000

acceptDetails( )
getSalary( )

1611

id	1102
job	Accountant
bsal	18000

acceptDetails( )
getSalary( )

1612

sal	18000
da	xxx
ta	xxx
emp	1612

```
PaySlip(Employee e) {
  e = 1612
}
```

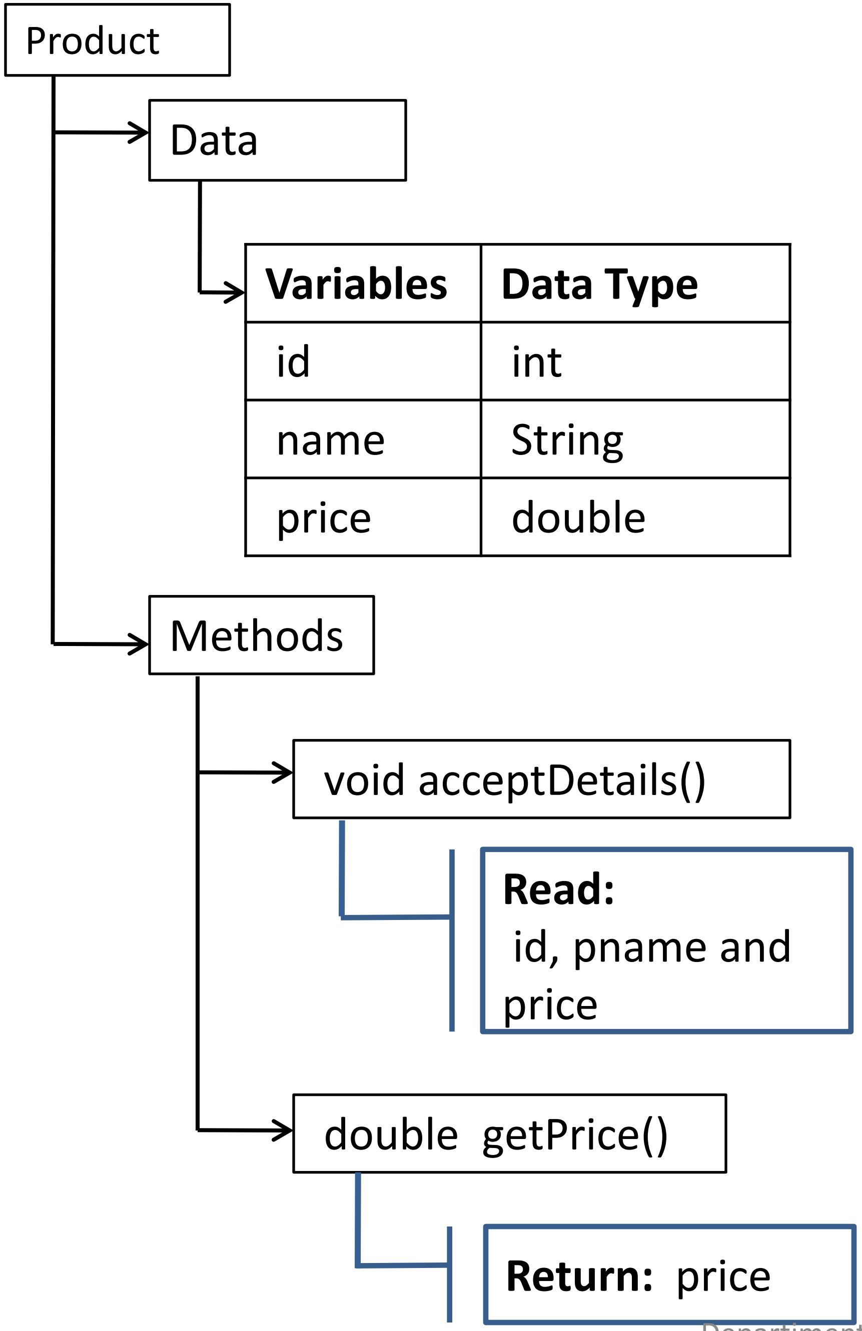
1613

1611
allen

1612
smith

1613
pay

Main block



Product

Data

Variables	Data Type
id	int
name	String
price	double

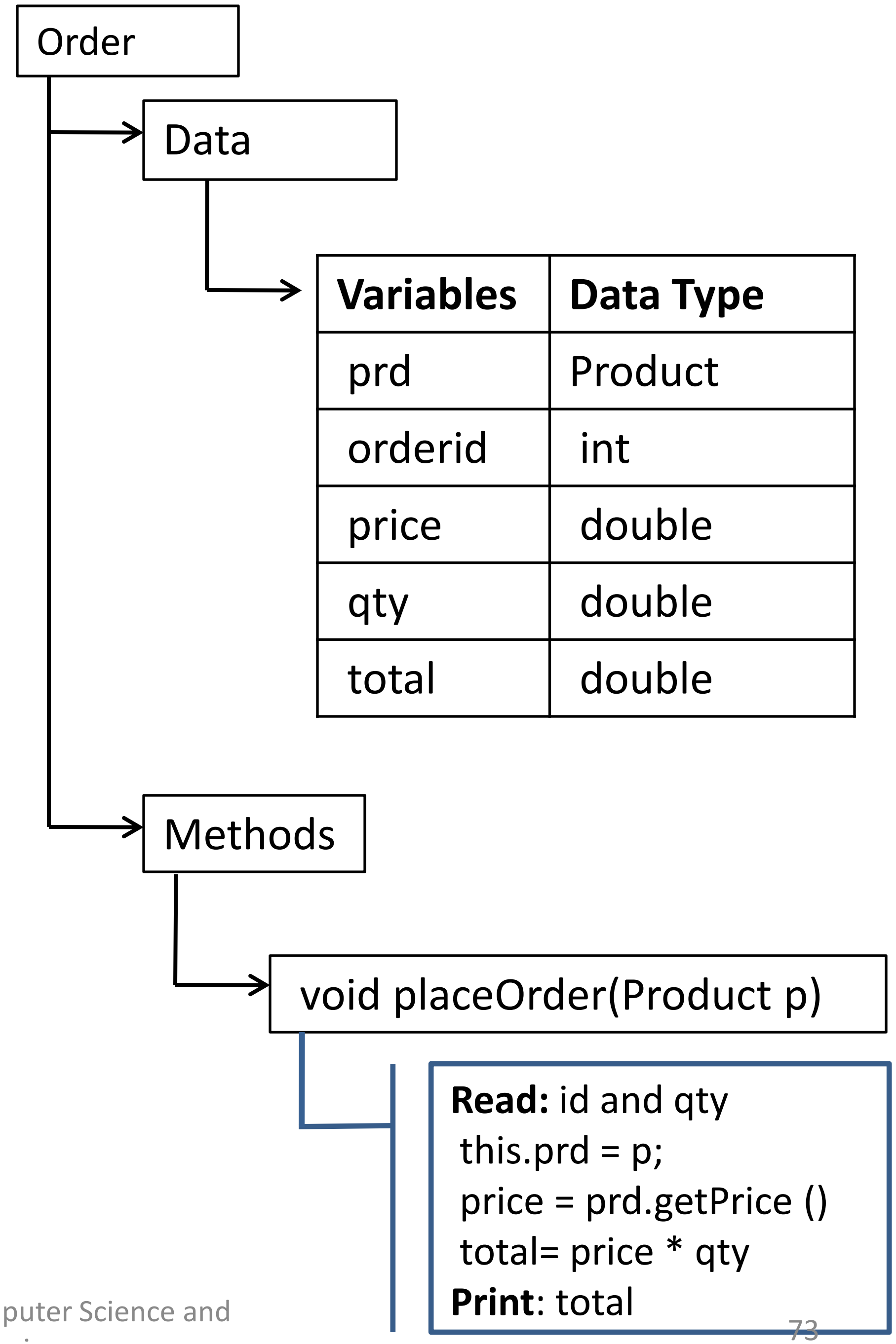
Methods

void acceptDetails()

Read:
id, pname and
price

double getPrice()

Return: price



Order

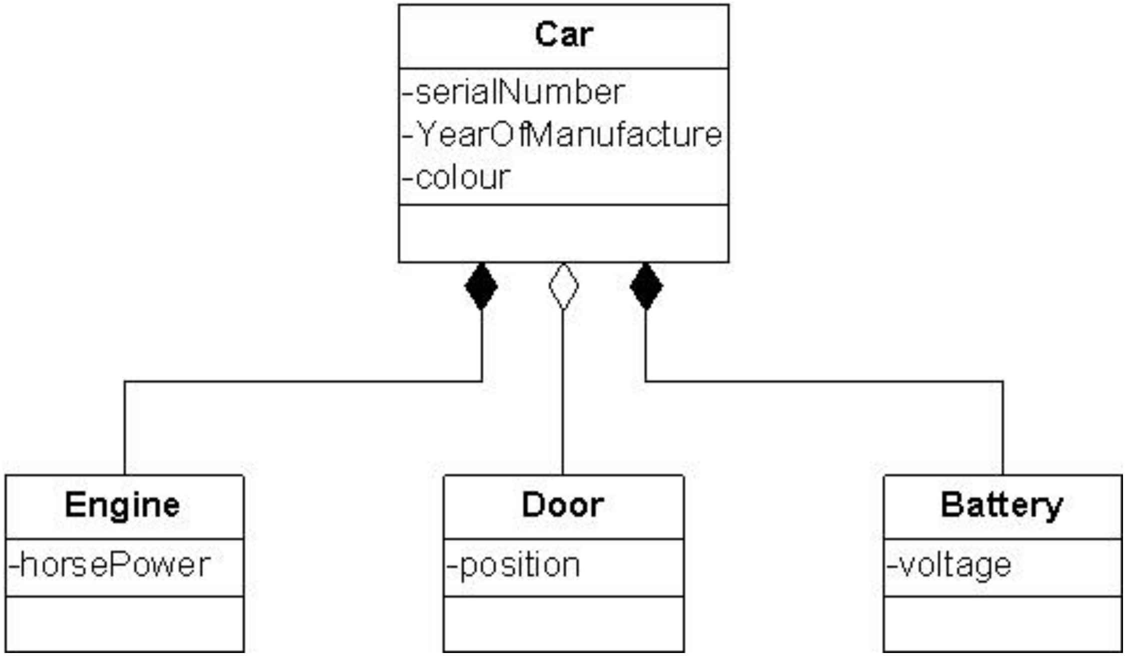
Data

Variables	Data Type
prd	Product
orderid	int
price	double
qty	double
total	double

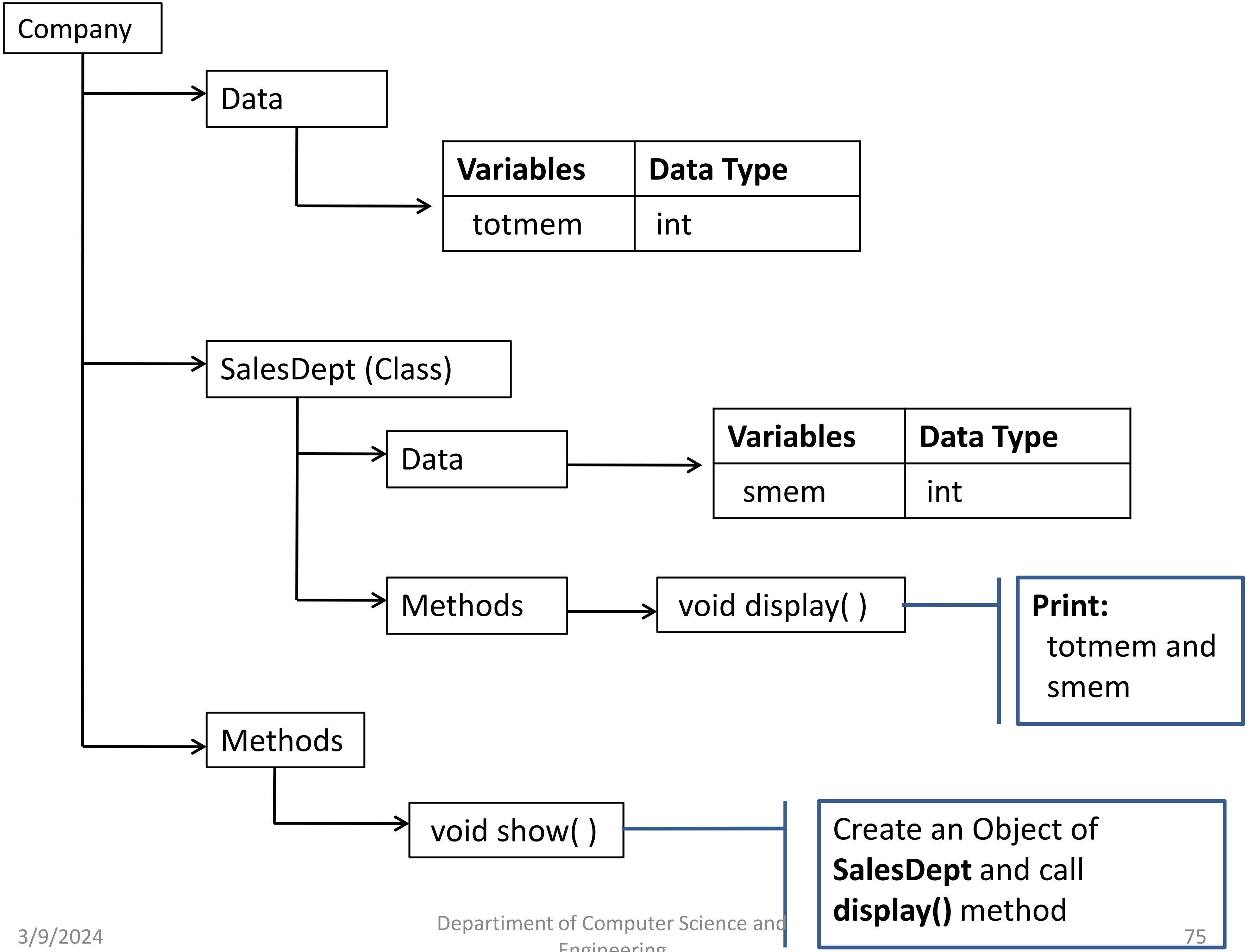
Methods

void placeOrder(Product p)

Read: id and qty
this.prd = p;
price = prd.getPrice ()
total= price * qty
Print: total



In the diagram above, the battery and the engine have no meaning outside of the car, as the car cannot work without either of them, so the relationship is formed using **composition**. However, a car can work without doors, so the relationship is formed using **aggregation**.



Company

Data

Variables	Data Type
totmem	int

SalesDept (Class)

Data

Variables	Data Type
smem	int

Methods

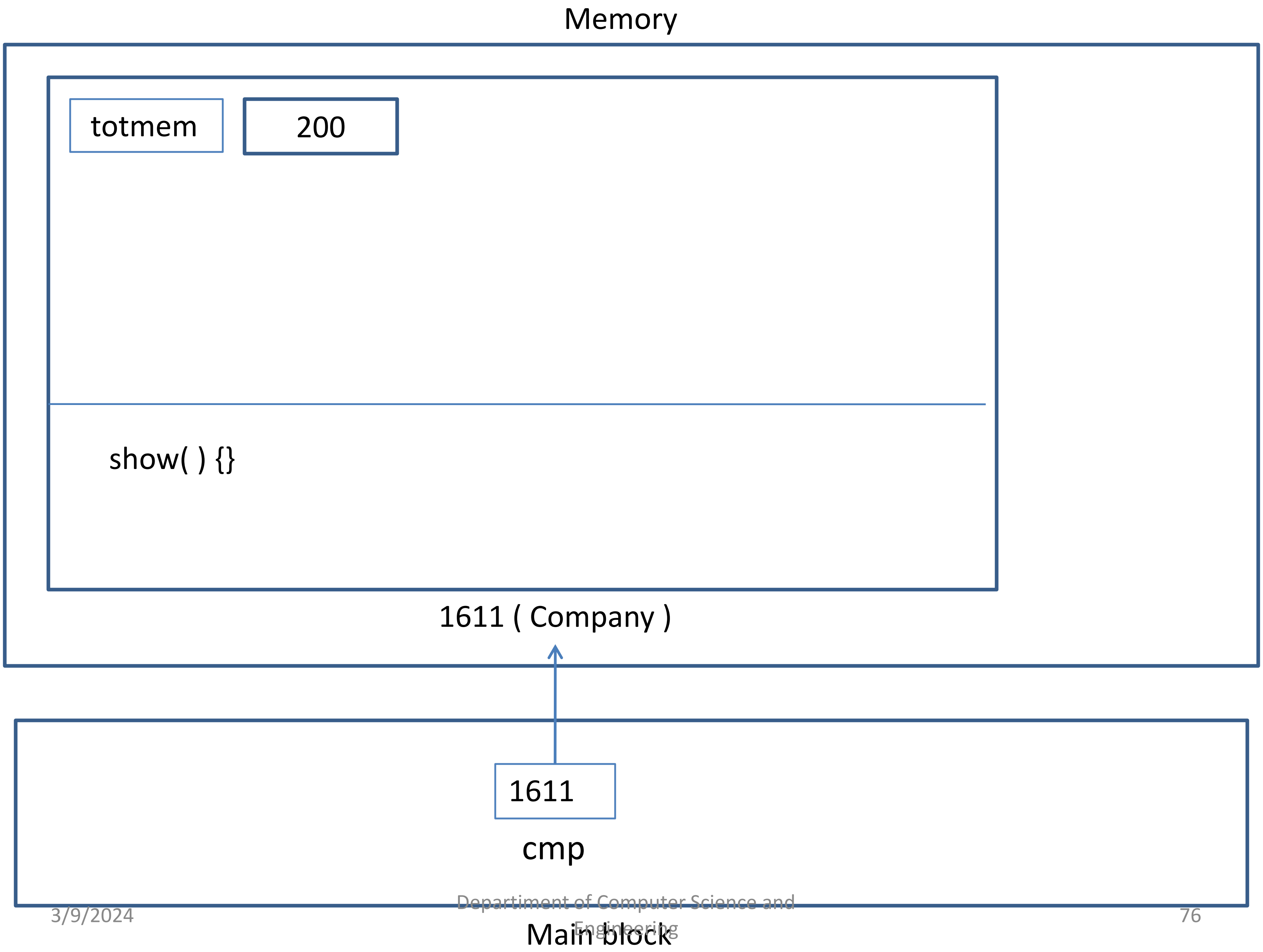
void display( )

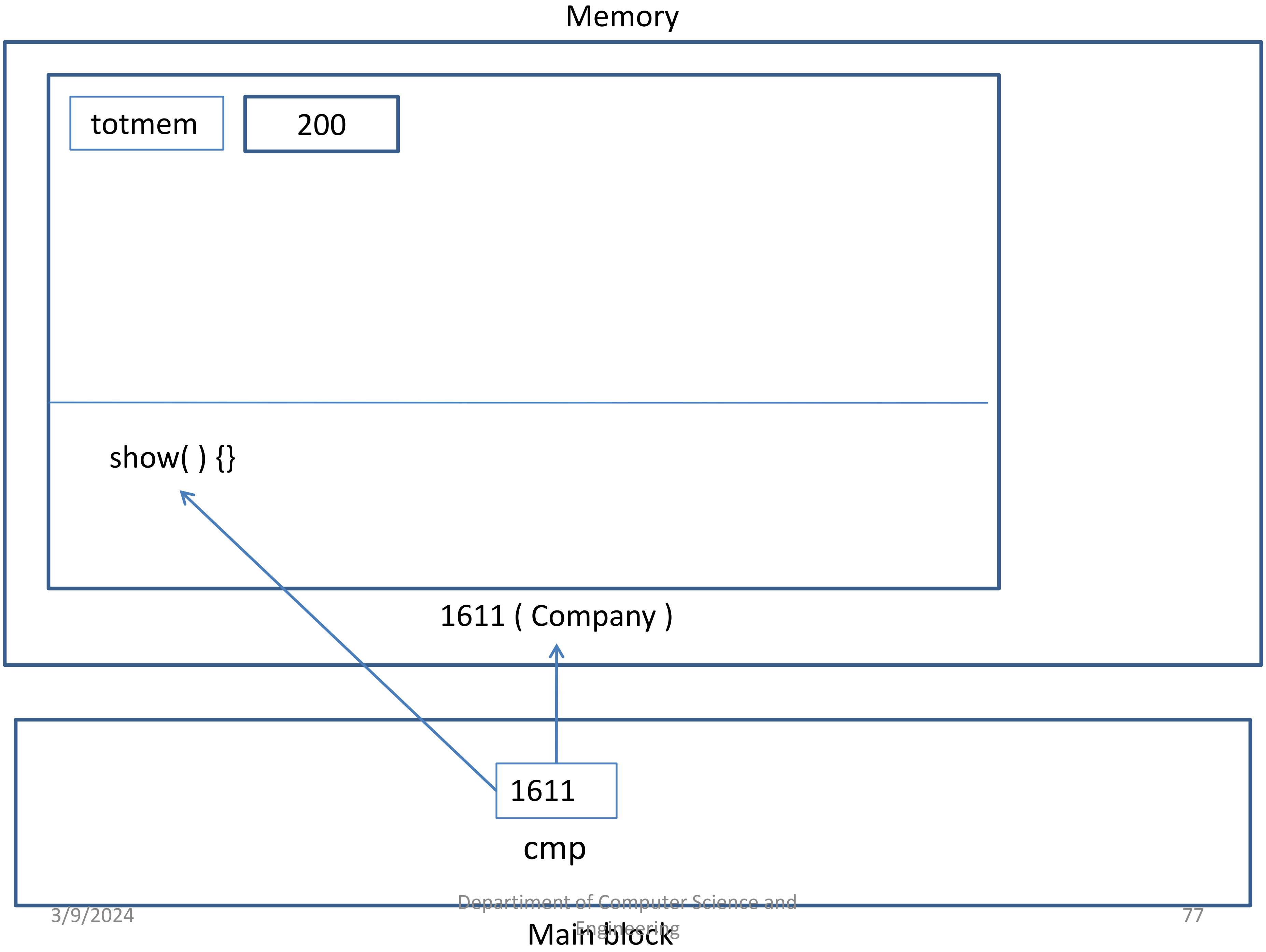
Print: totmem and smem

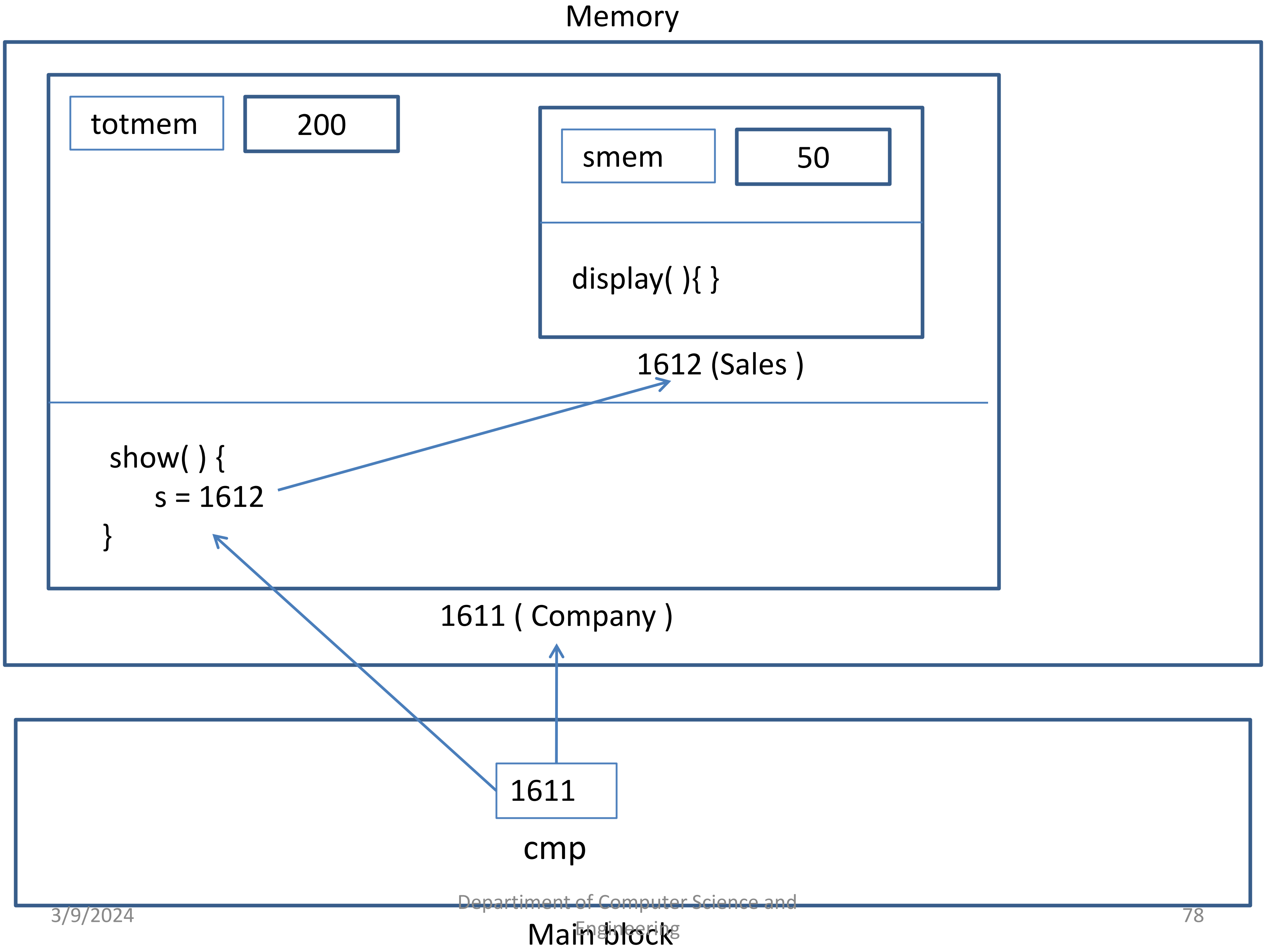
Methods

void show( )

Create an Object of **SalesDept** and call **display()** method


Memory
totmem
200
show( ) {}
1611 (Company)
1611
cmp
Main block


Memory
totmem
200
show( ) {}
1611 (Company)
1611
cmp
Main block



Memory

totmem

200

smem

50

display( ){ }

1612 (Sales)

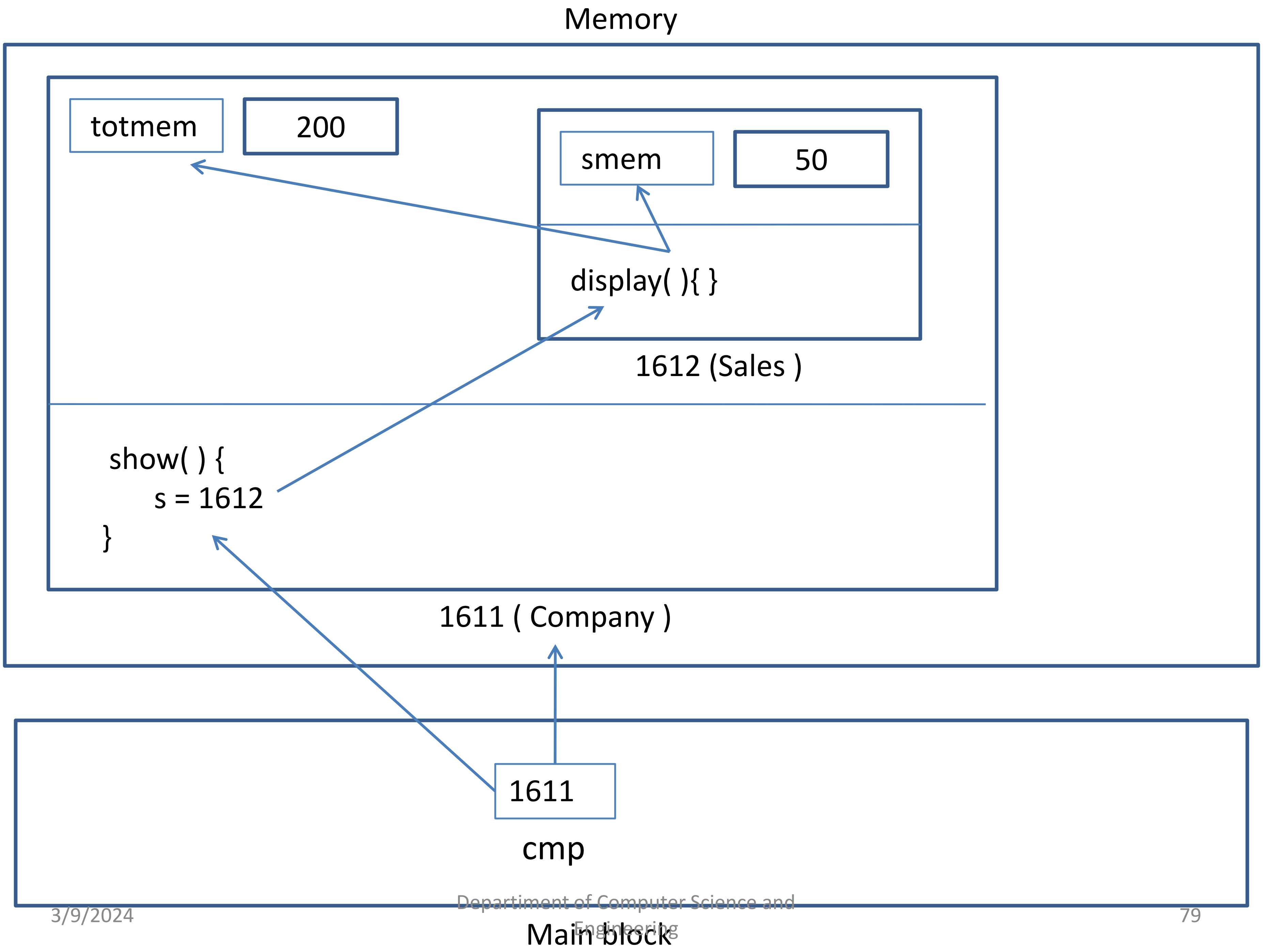
show( ) {
s = 1612
}

1611 (Company)

1611

cmp

Main block



Memory

totmem 200

smem 50

display( ){ }

1612 (Sales)

```
show( ) {
      s = 1612
}
```

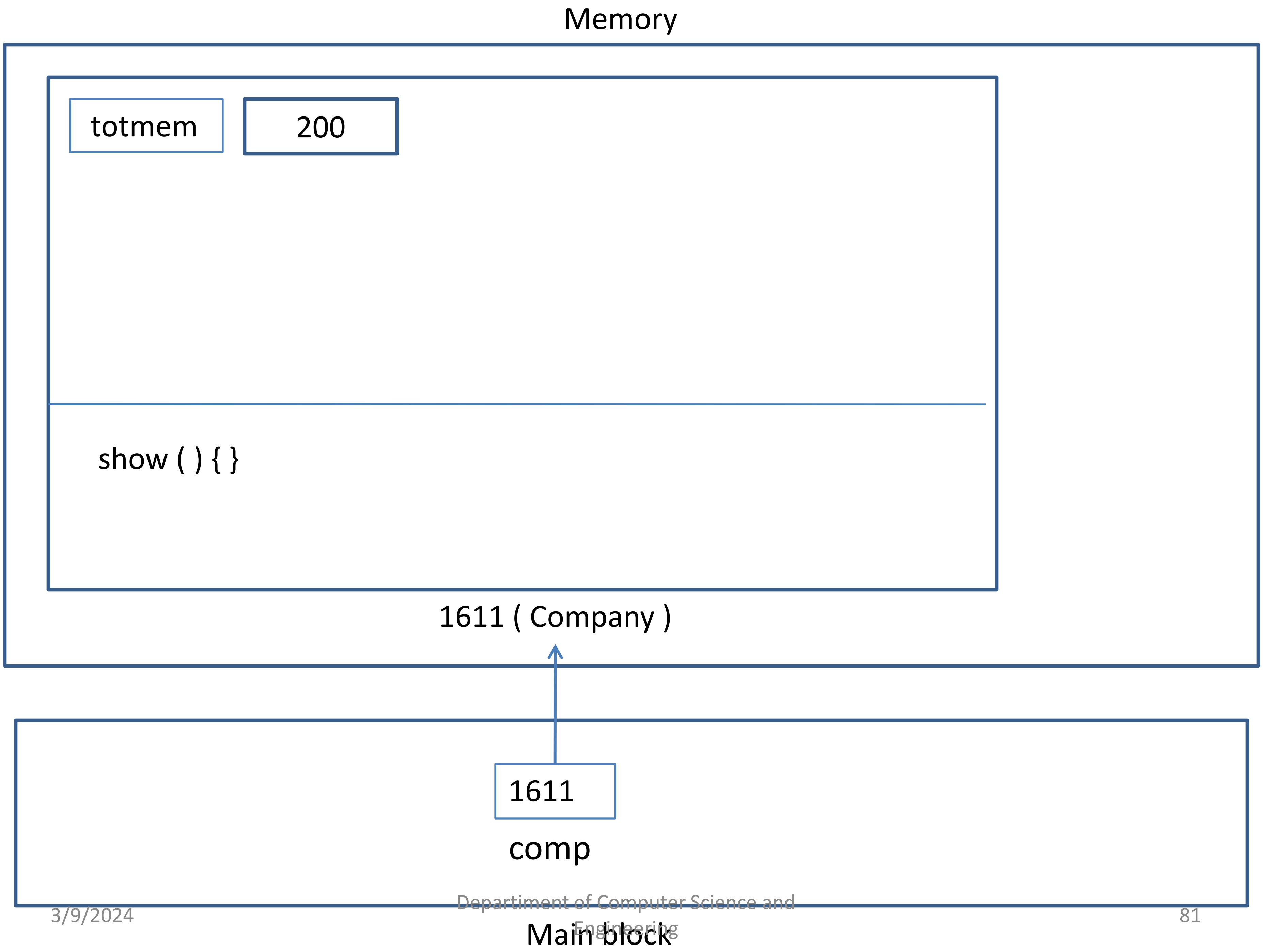
1611 (Company)

1611

cmp

Main block

Using Static Class



Memory

totmem

200

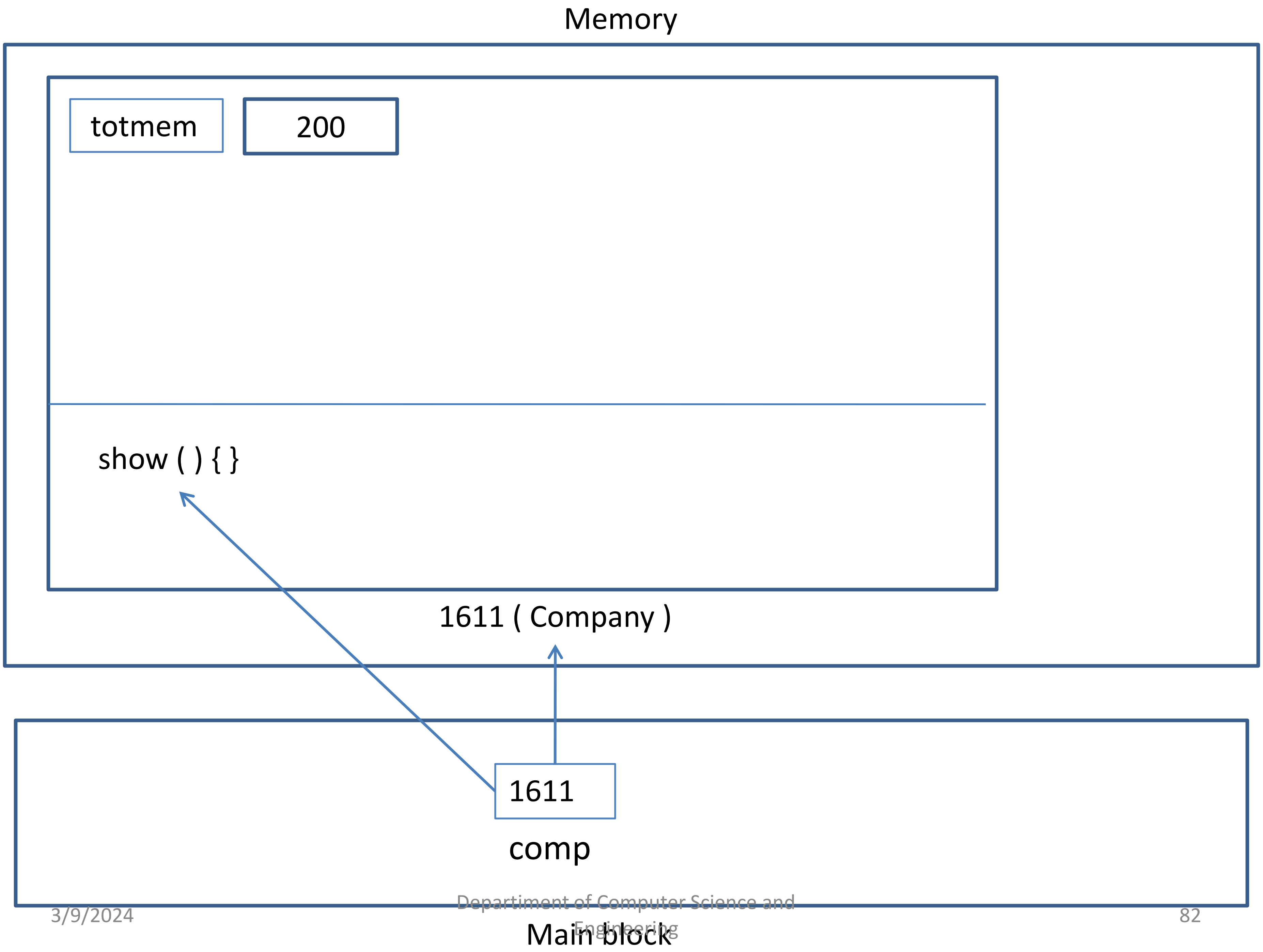
show ( ) { }

1611 (Company)

1611

comp

Main block



Memory

totmem

200

show ( ) { }

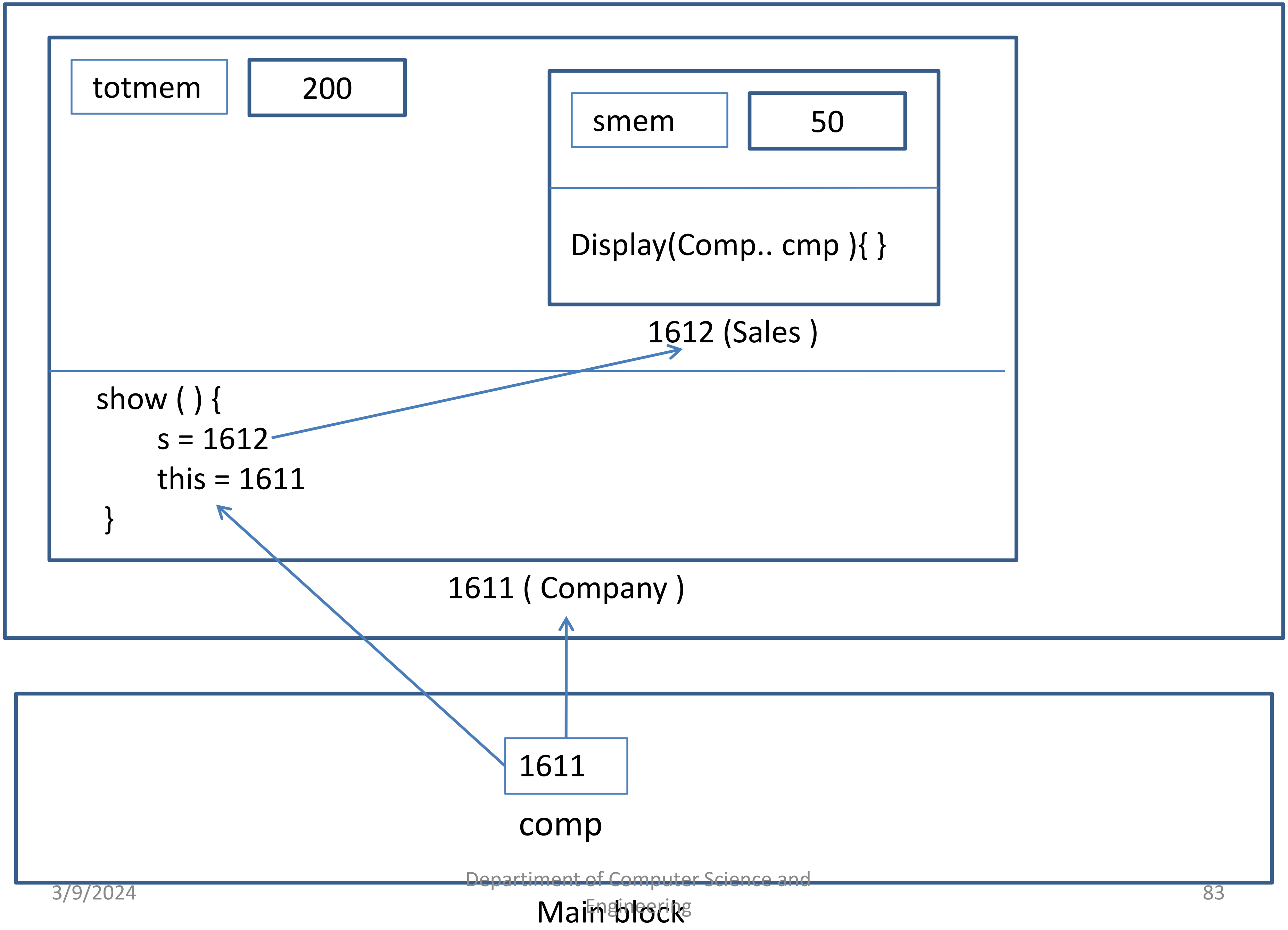
1611 (Company)

1611

comp

Main block

Memory



totmem

200

smem

50

Display(Comp.. cmp){ }

1612 (Sales)

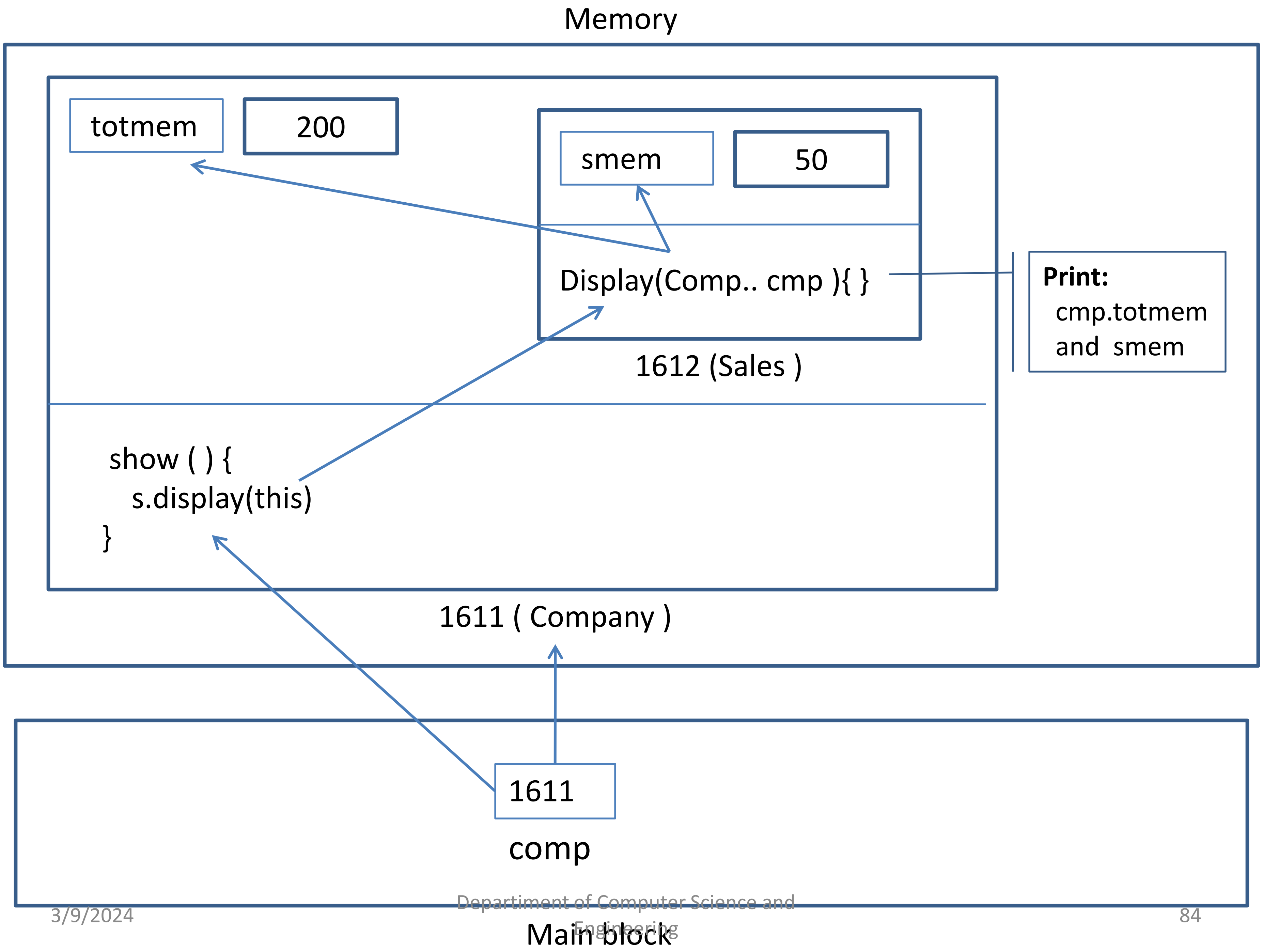
```
show ( ) {
        s = 1612
        this = 1611
 }
```

1611 (Company)

1611

comp

Main block



Memory

totmem

200

smem

50

Display(Comp.. cmp){ }

1612 (Sales)

Print:
cmp.totmem
and smem

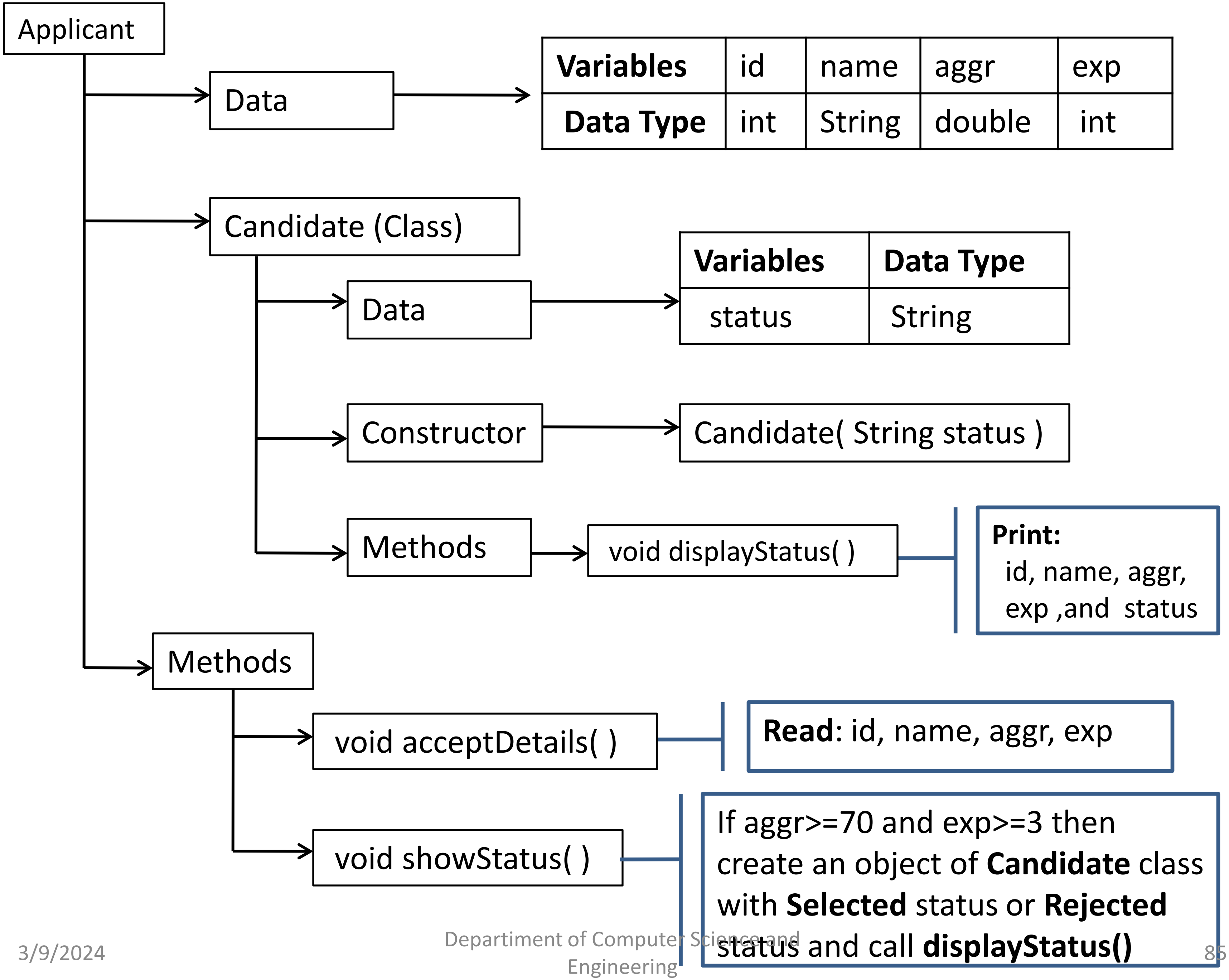
show ( ) {
s.display(this)
}

1611 (Company)

1611

comp

Main block



Applicant

- Data

Variables	id	name	aggr	exp
Data Type	int	String	double	int

- Candidate (Class)
  - Data

Variables	Data Type
status	String

  - Constructor → Candidate(String status)
  - Methods → void displayStatus( )
    - **Print:** id, name, aggr, exp ,and status
- Methods
  - void acceptDetails( )
    - **Read**: id, name, aggr, exp
  - void showStatus( )
    - If aggr>=70 and exp>=3 then create an object of **Candidate** class with **Selected** status or **Rejected** status and call **displayStatus()**

Using Inheritance

Inheritance

A class, which inherits the properties from another class.

Superclass

- A Superclass / Base class is a class that has been inherited by another class. It allows the inheriting class to inherit its state and behaviors.

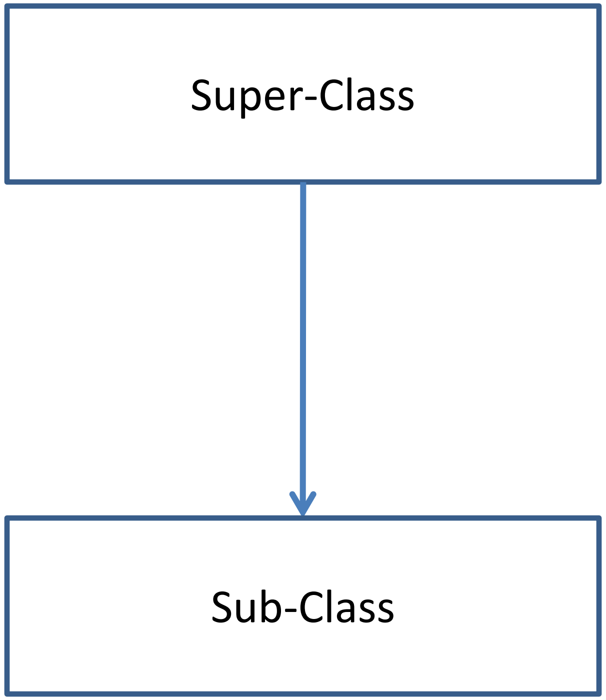
Subclass

- A Subclass / Derived class is a class that inherits the member variables and member functions from another class.

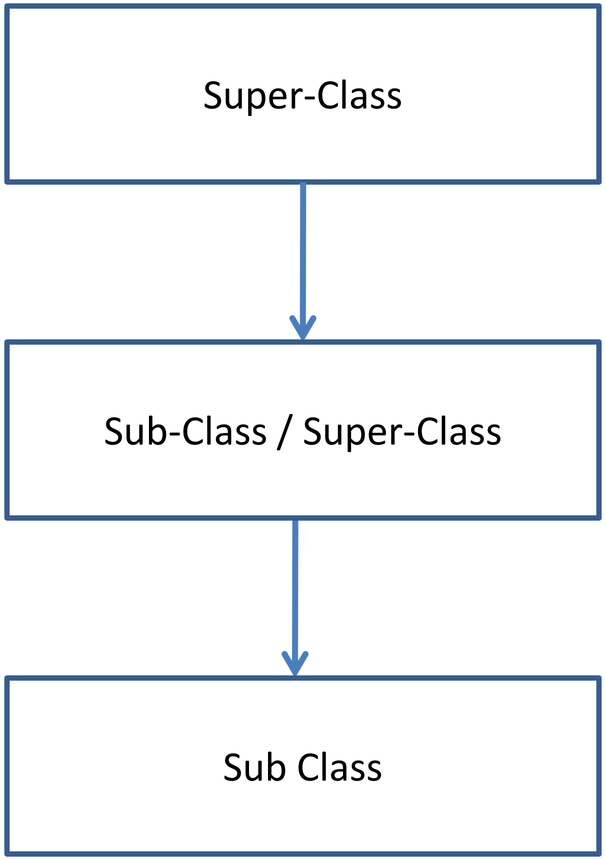
Inheritance offers Re-Usability

✍ : The keyword **extends** is used to implement inheritance

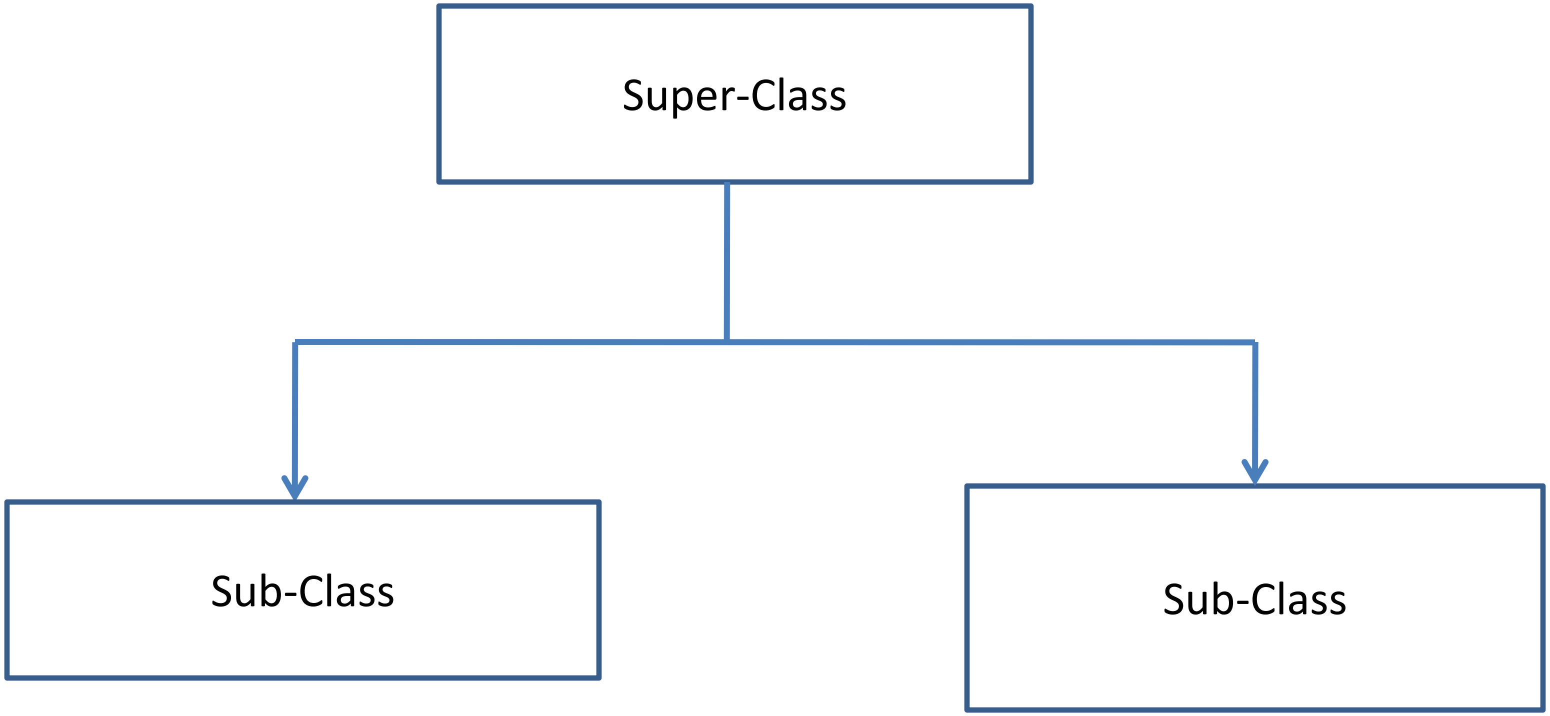
Single Inheritance



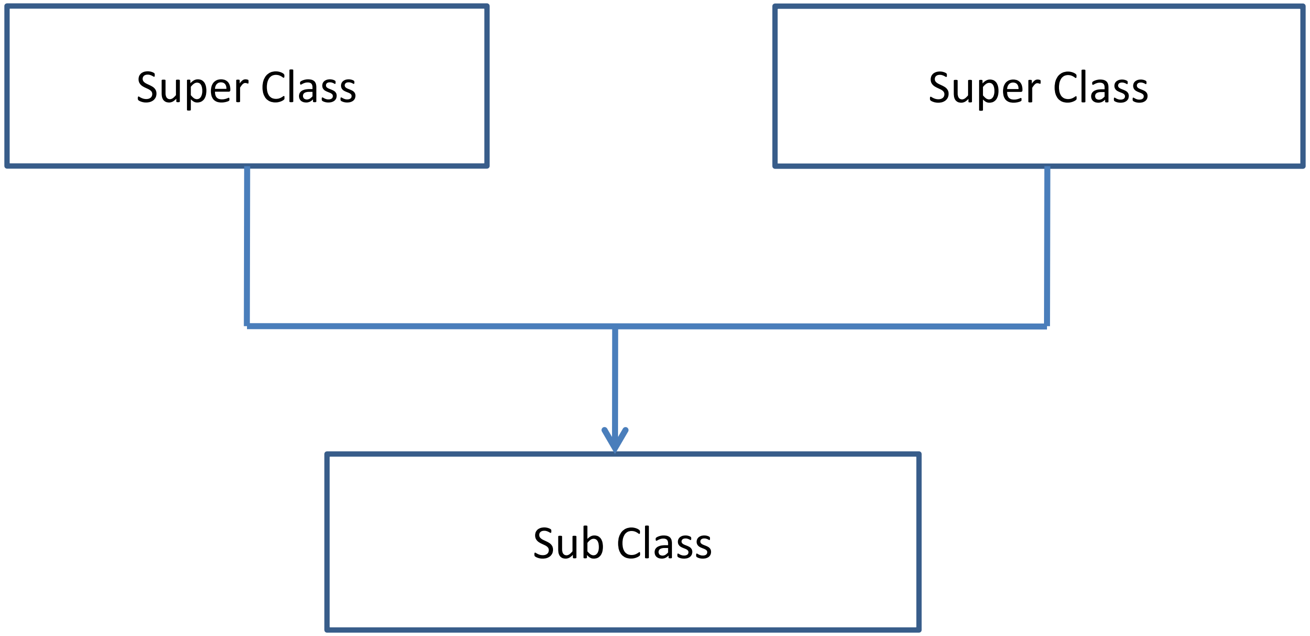
Multilevel Inheritance



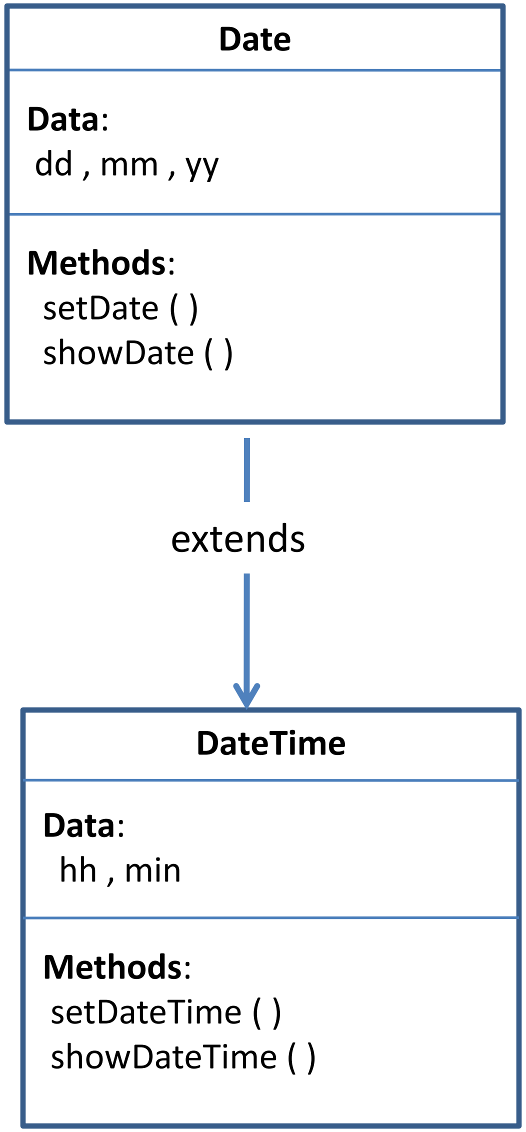
Hierarchical Inheritance



Multiple Inheritance



✍ : Multiple Inheritance is not supported by Java directly. However, you can accomplish this by using **interfaces**



Date

Data:
dd , mm , yy

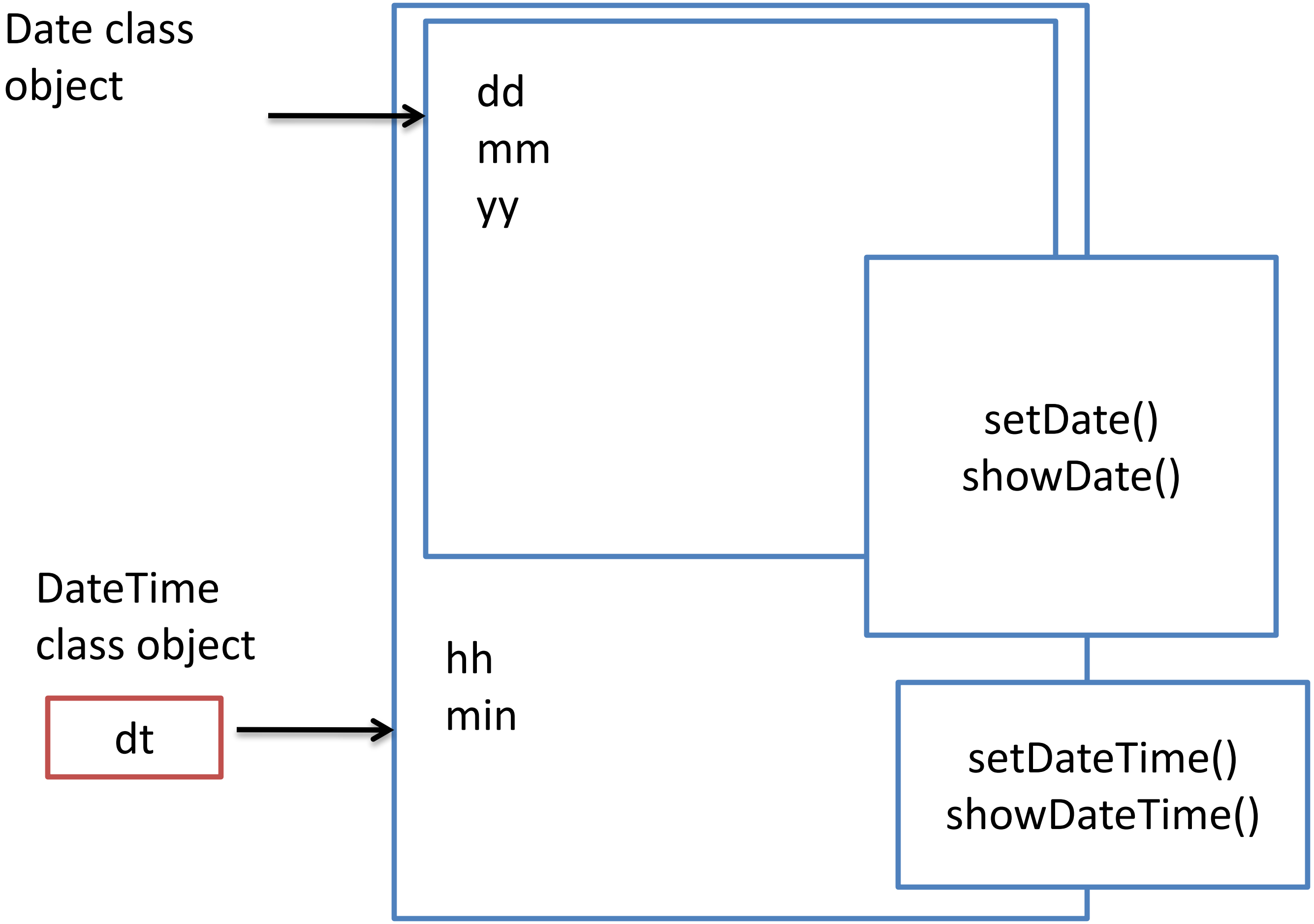
Methods:
setDate ( )
showDate ( )

extends

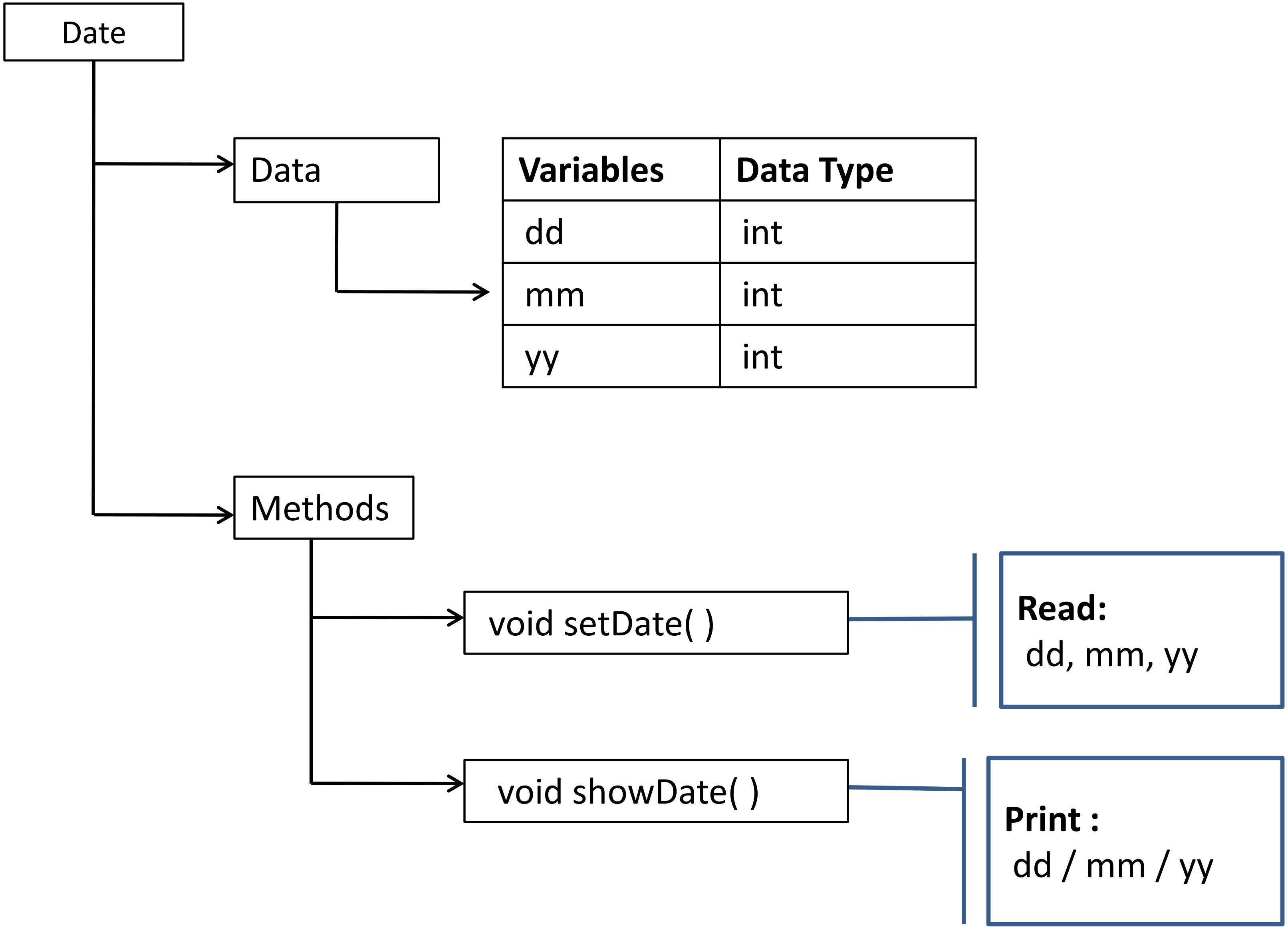
DateTime

Data:
hh , min

Methods:
setDateTime ( )
showDateTime ( )



DateTime class objects contains the copy of Date class



Date

Data

Variables	Data Type
dd	int
mm	int
yy	int

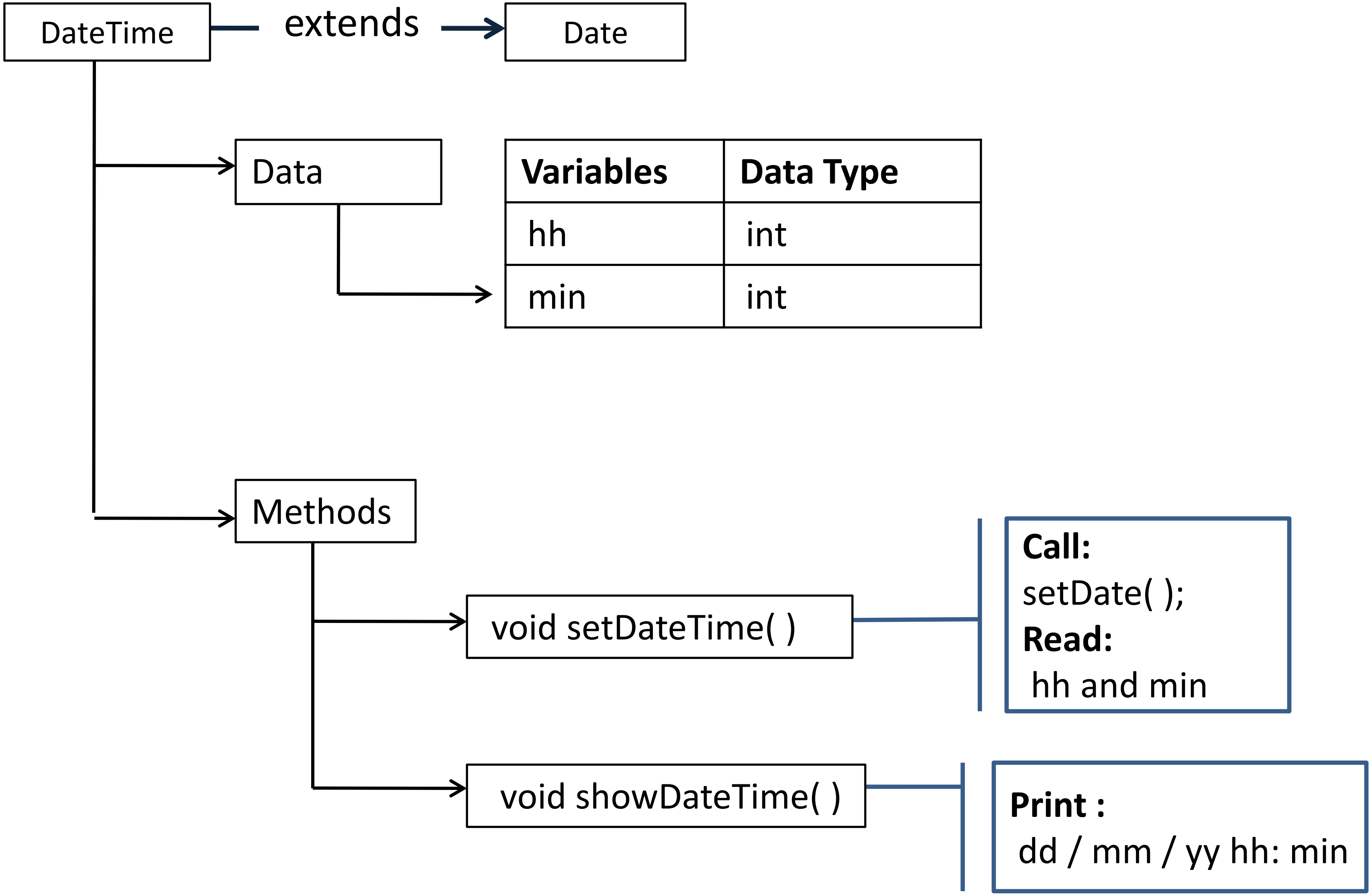
Methods

void setDate( )

Read:
dd, mm, yy

void showDate( )

Print :
dd / mm / yy



DateTime extends Date

Data

Variables	Data Type
hh	int
min	int

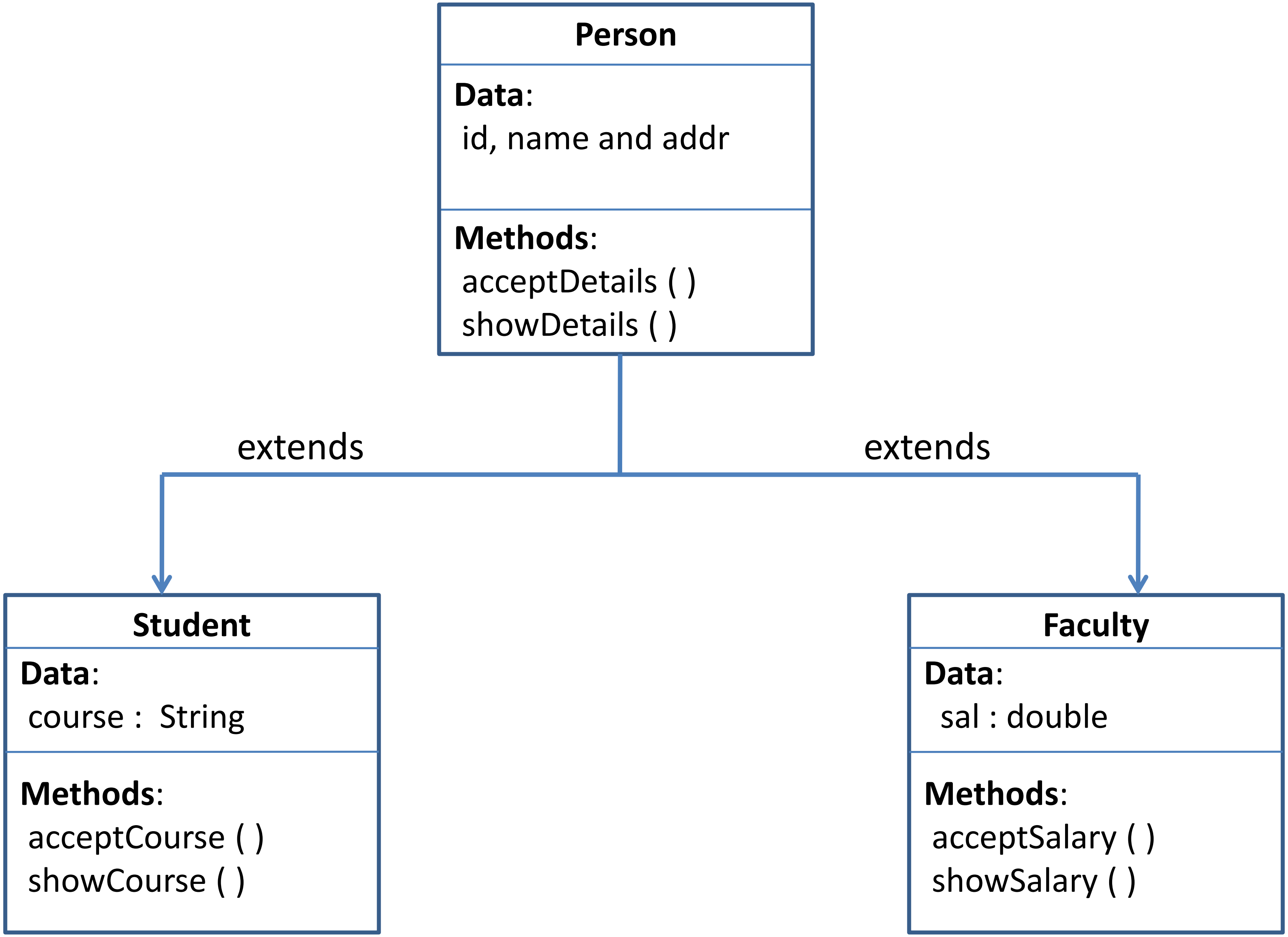
Methods

void setDateTime( )

Call:
setDate( );
Read:
hh and min

void showDateTime( )

Print :
dd / mm / yy hh: min



Person

Data:
id, name and addr

Methods:
acceptDetails ( )
showDetails ( )

extends

extends

Student

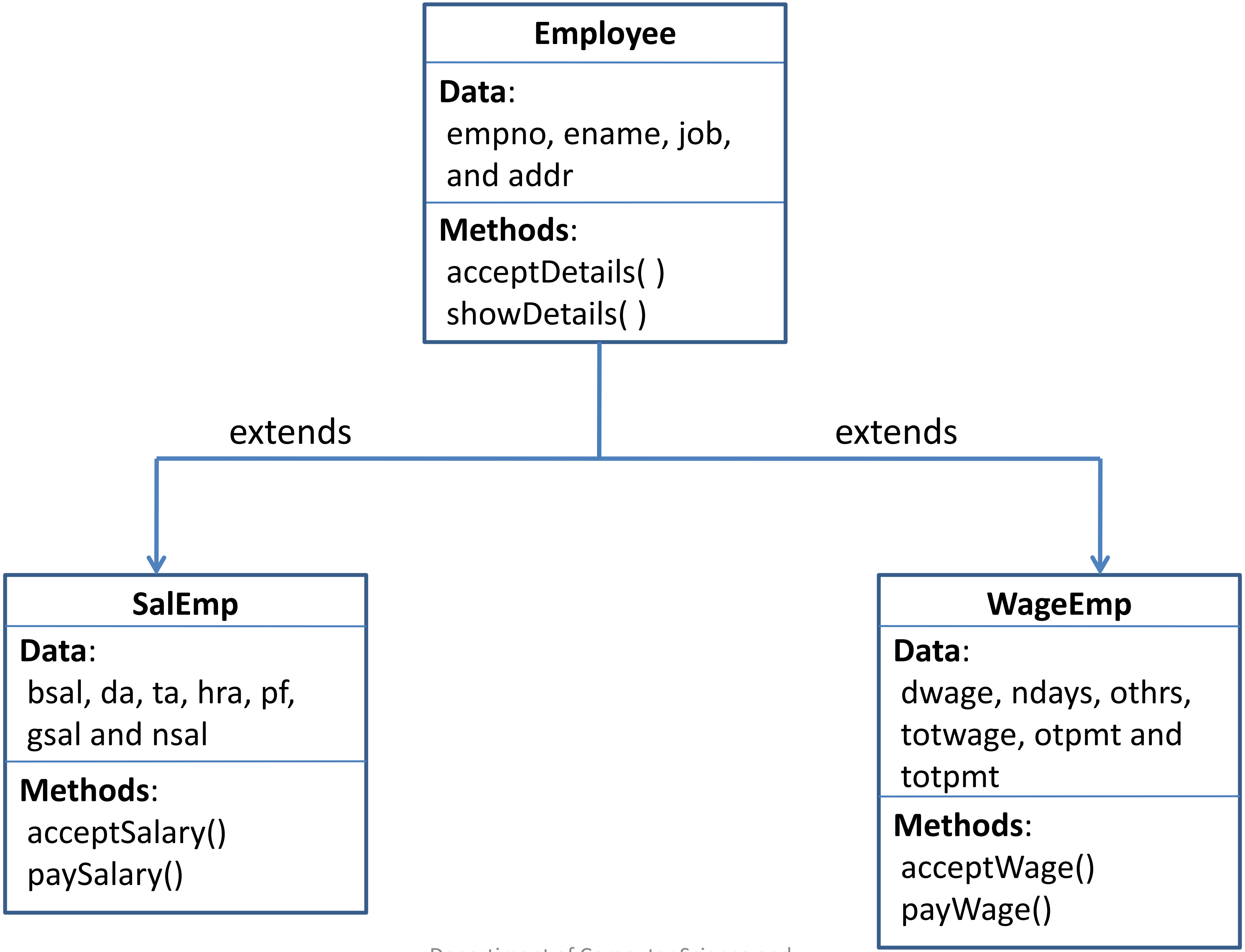
Data:
course : String

Methods:
acceptCourse ( )
showCourse ( )

Faculty

Data:
sal : double

Methods:
acceptSalary ( )
showSalary ( )



Employee

Data:
empno, ename, job,
and addr

Methods:
acceptDetails( )
showDetails( )

extends

extends

SalEmp

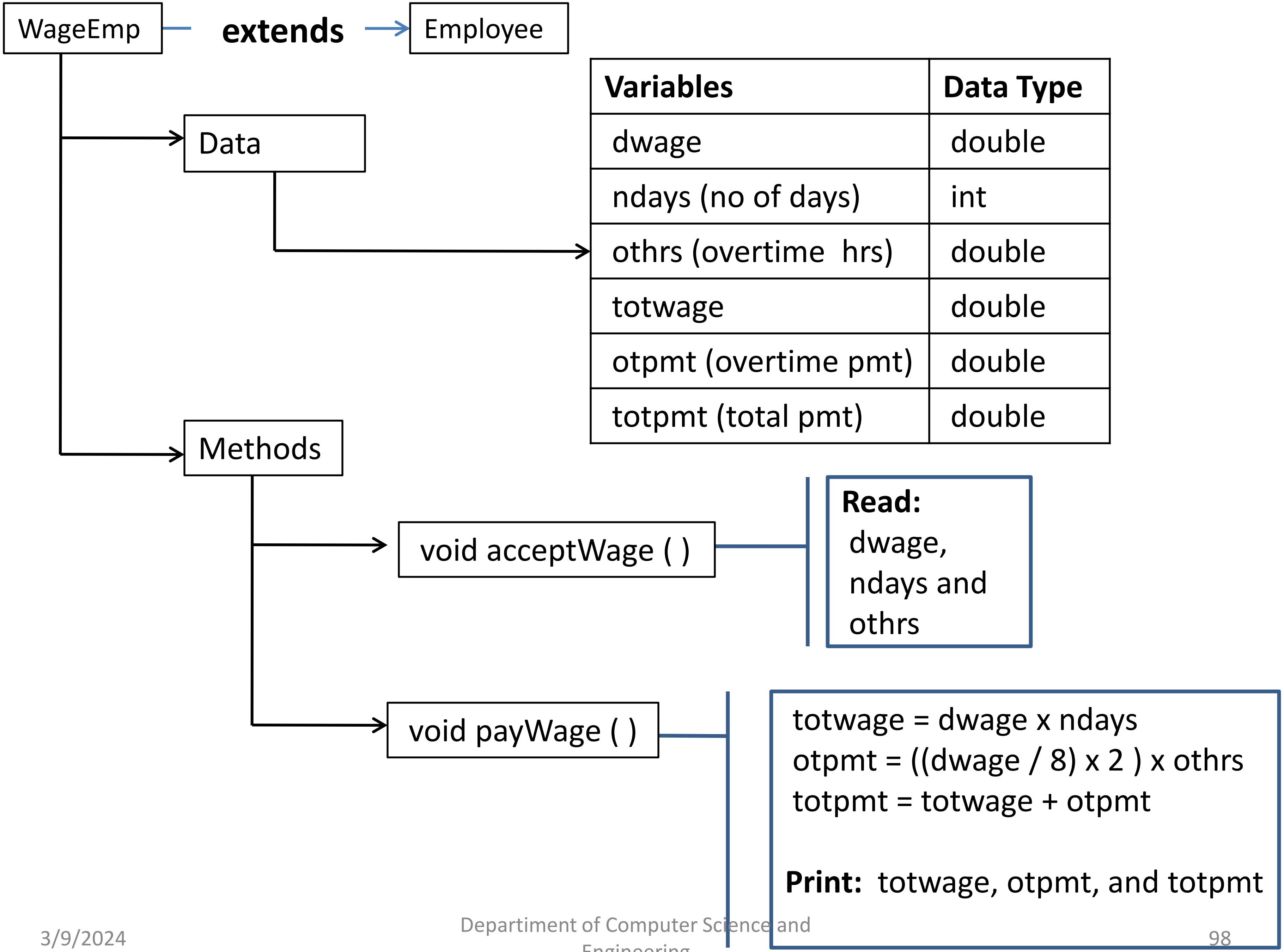
Data:
bsal, da, ta, hra, pf,
gsal and nsal

Methods:
acceptSalary()
paySalary()

WageEmp

Data:
dwage, ndays, othrs,
totwage, otpmt and
totpmt

Methods:
acceptWage()
payWage()



WageEmp **extends** Employee

Data

Variables	Data Type
dwage	double
ndays (no of days)	int
othrs (overtime hrs)	double
totwage	double
otpmt (overtime pmt)	double
totpmt (total pmt)	double

Methods

void acceptWage ( )

Read:
dwage,
ndays and
othrs

void payWage ( )

totwage = dwage x ndays
otpmt = ((dwage / 8) x 2) x othrs
totpmt = totwage + otpmt

Print: totwage, otpmt, and totpmt