CLASS & OBJECT

Class

```
public class MyHelloWorld{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                int x = sc.nextInt();
                System.out.println(x);
        }
}
```

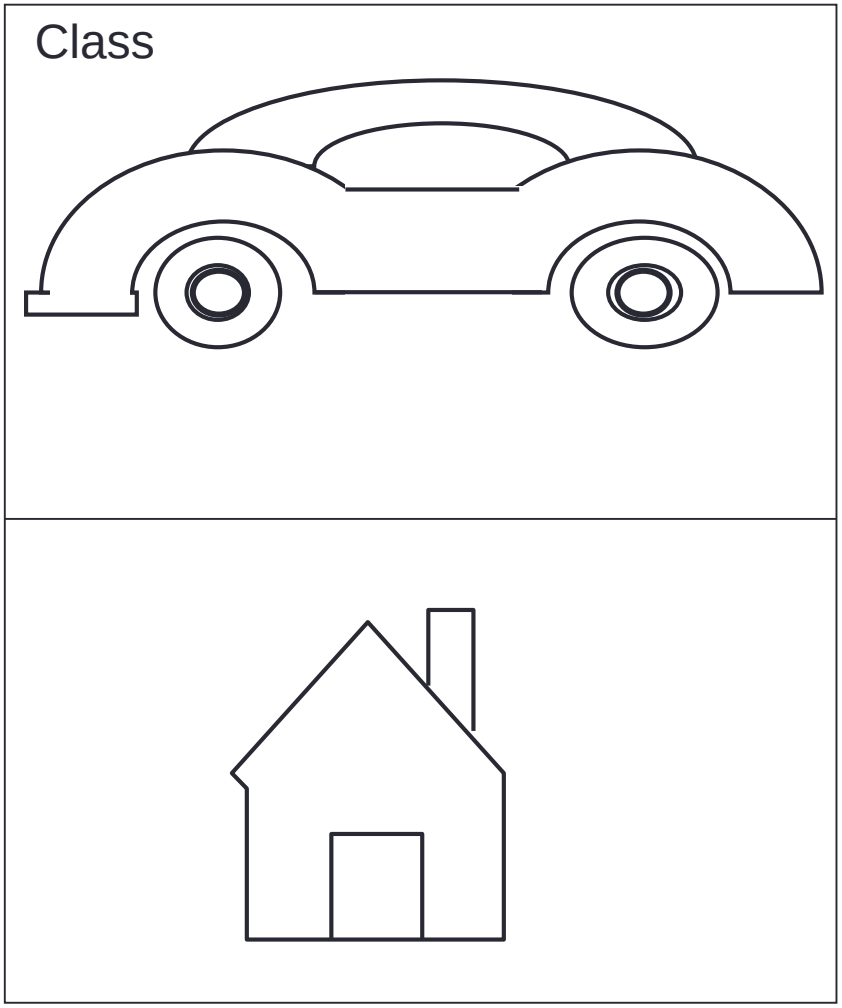
Class Types

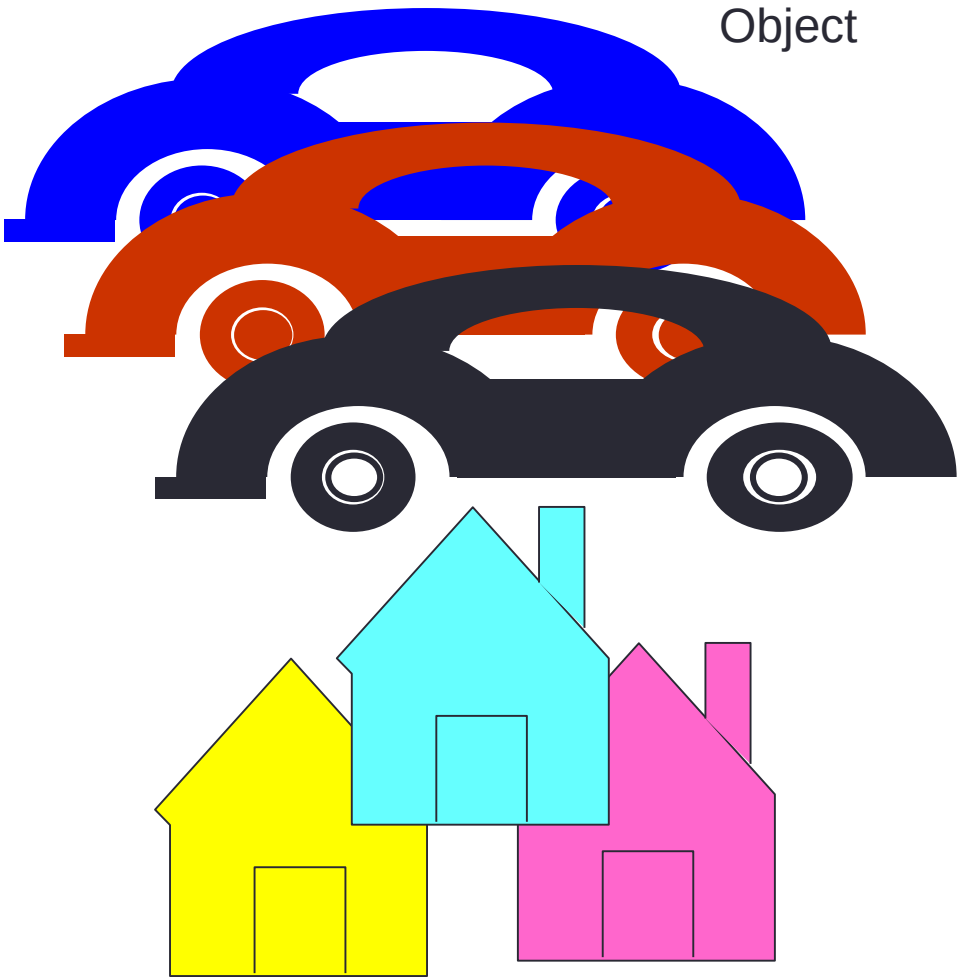
- **Driver class** – often just the class that contains a main
- **Boundary class** (service class or utility class) – a class that collects the related static methods and no data such as the Math class.
- **Entity class** – a class that represent any object in the real world such as Student, Employee, BankAccount, Circle, there are both data and methods.

Class & Object

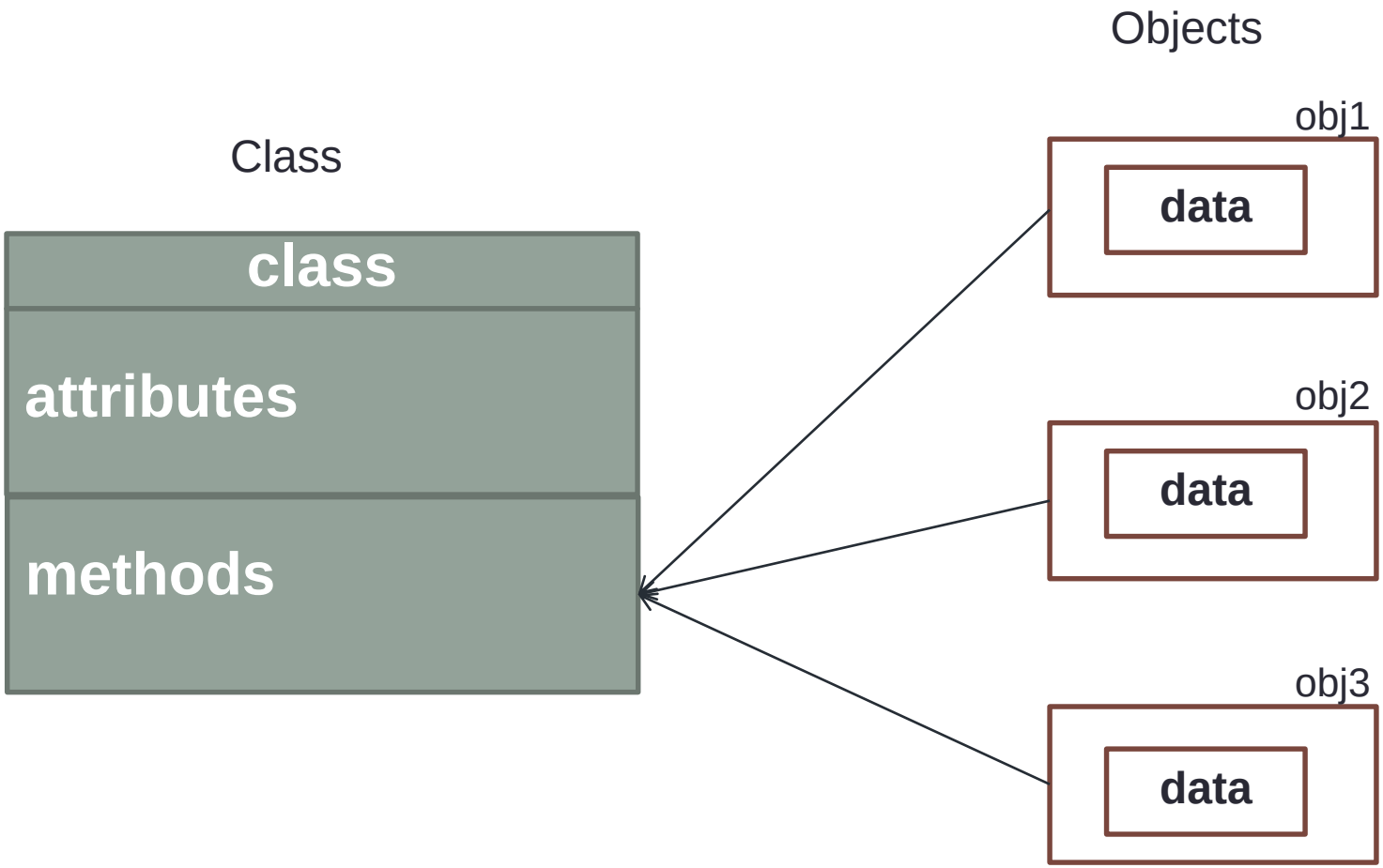
- A class is a template or blueprint for creating an object.
- A class defines what types of data are included in the object and specifies the operations the object performs.
- Object is created from a class.
- Creating an object from a class is known as instantiating an object.

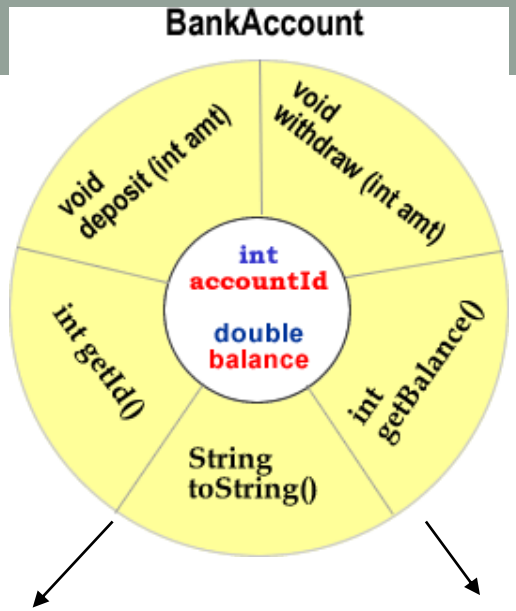
Class & Object





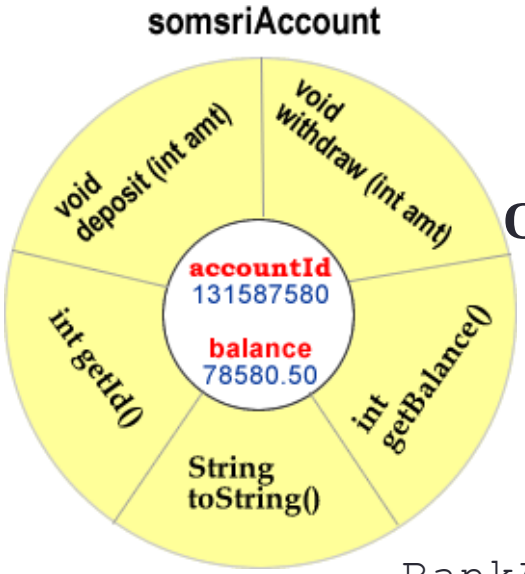
Class & Object



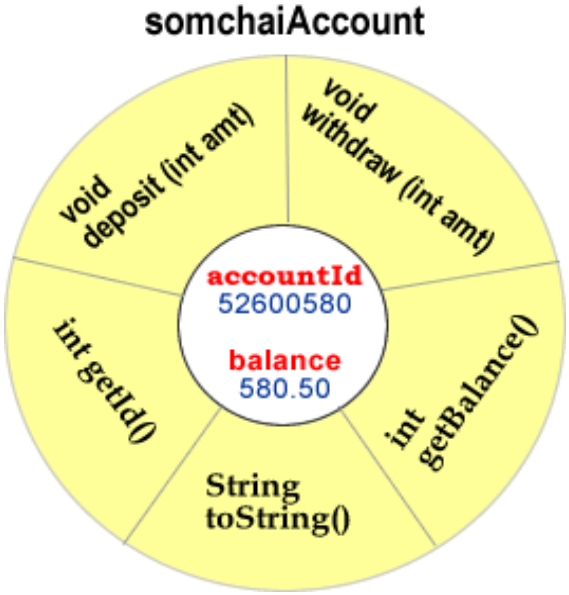


Class

```
BankAccount somsriAccount;
somsriAccount = new BankAccount();
```



Objects/Instances




```
BankAccount somchaiAccount= new BankAccount();
```

```
public class BankAccount{
 private int accId;
 private double balance;
```

Attributes

```
 public void setAccId(int id){
        accId=id;
 }
 public void setBalance(double bal){
        balance=bal;
 }
 public int getAccId(){
        return accId;
 }
 public double getBalance(){
        return balance;
 }
 public void deposit(double amount){
        balance=balance+amount;
 }
 public void withdraw(double amount){
        if(amount<=balance)
           balance=balance-amount;
        else
           System.out.println("cannot withdraw please check your balance")
 }
 public String toString(){
        return "Account Id = "+ accId +"\nBalance = " +balance;
  }
}
```

Methods

Creating Objects

- A variable holds either a *primitive type* or a *reference* to an object
- A class name can be used as a type to declare an *object reference variable*

```
BankAccount somsakAccount ;
```

- No object is created with this declaration
- An object reference variable holds the address of an object
- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object

```
somsakAccount = new BankAccount();
```

Creating an object is called *instantiation*

An object is an *instance* of a particular class

Creating Object

- Object Declaration

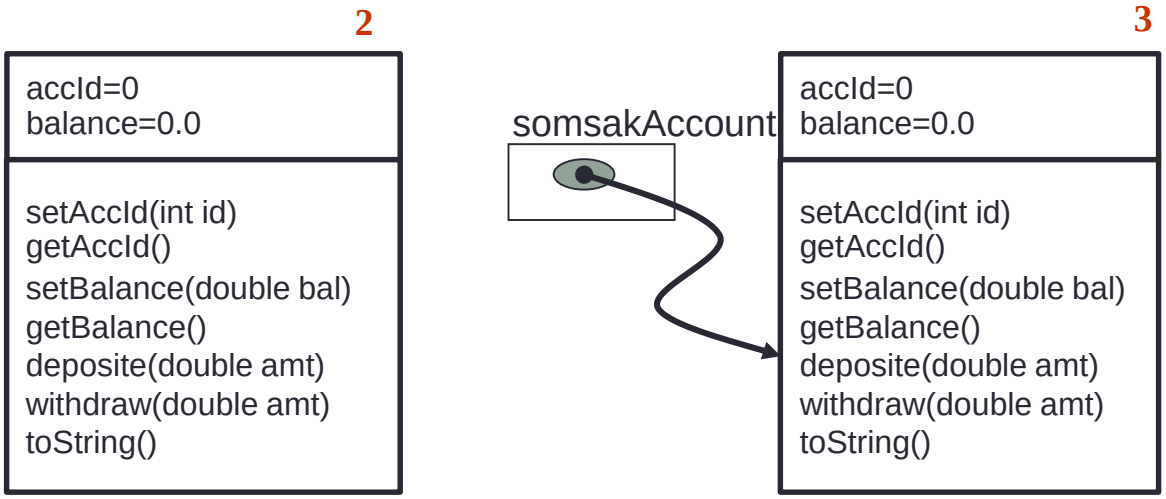
  1

  BankAccount somsakAccount;

- Object construction (Instantiation)

  3 2

  somsakAccount =new BankAccount();



Driver Programs

- A driver program drives the use of other, more interesting parts of a program
- Driver programs are often used to test other parts of the software

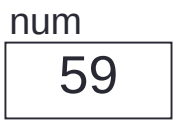
TestBankAccount.java

```
public class TestBankAccount{
 public static void main(String args[]){
      BankAccount somchaiAccount=new BankAccount();
      somchaiAccount.setAccId(123456);
      somchaiAccount.deposit(500);
      somchaiAccount.withdraw(100);
      System.out.println(somchaiAccount.getBalance());
      System.out.println(somchaiAccount);
 }
}
```

Primitive and Object Variables

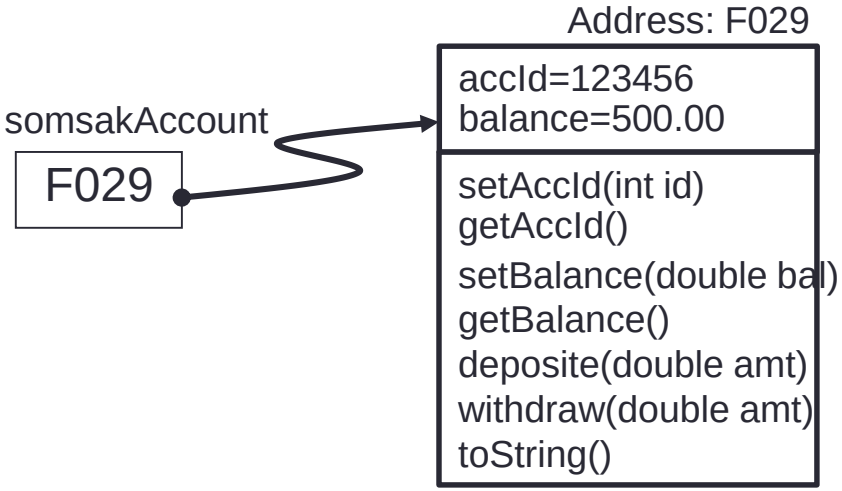
- A primitive variable
  - boolean, char, byte short, int, long, float and double

    E.g. `int num=59;`



- A reference variable

  E.g. `BankAccount somsakAccount= new BankAccount();`

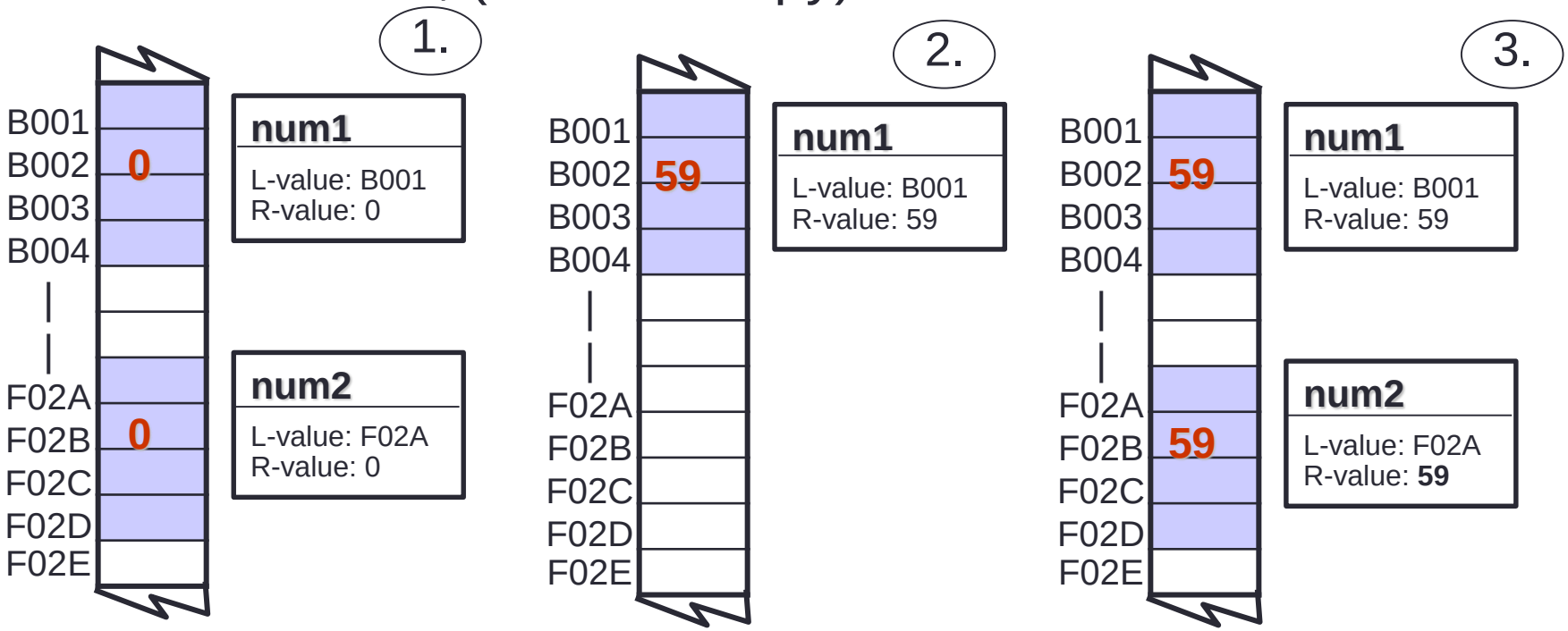


Primitive & Reference Variables

- Each variable has its own L-value and R-value
  1. The L-value is its address
  2. The R-value is its value

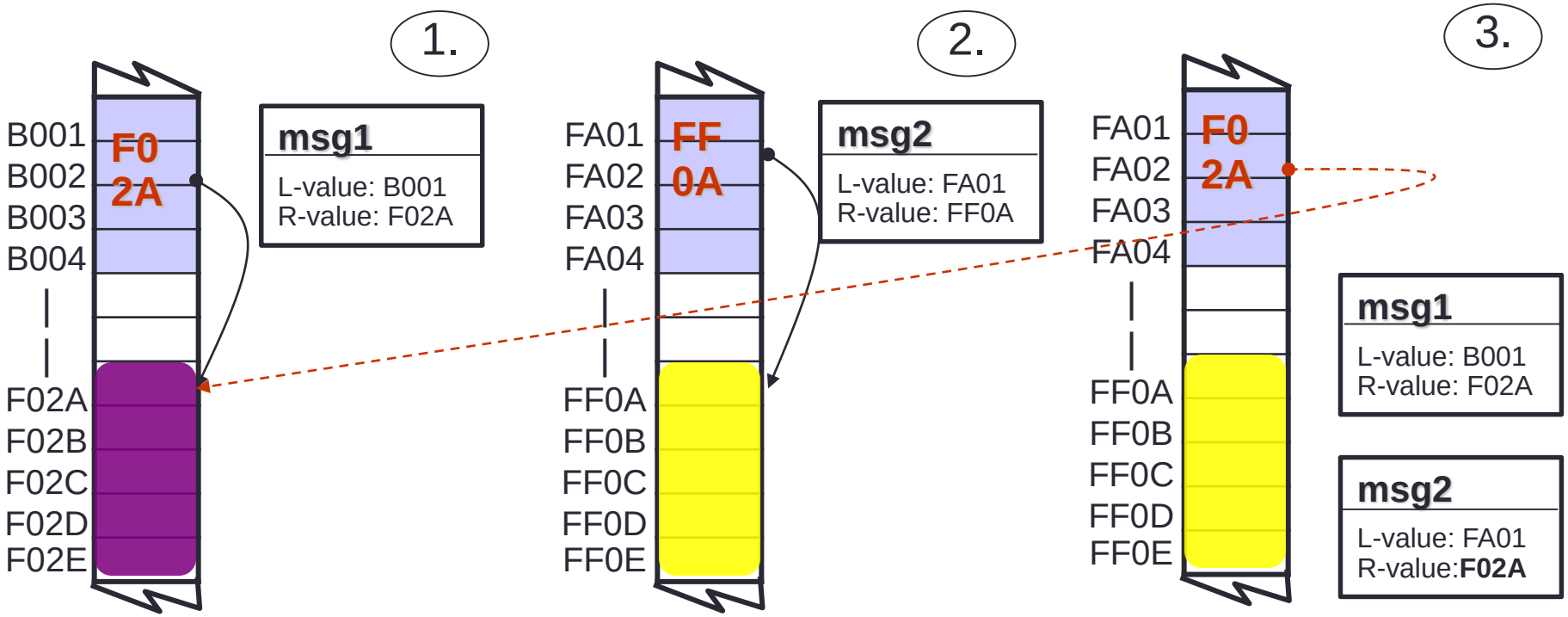
Primitive Assignment

1. int num1, int num2;
2. num1=59;
3. num2=num1; (R-value Copy)



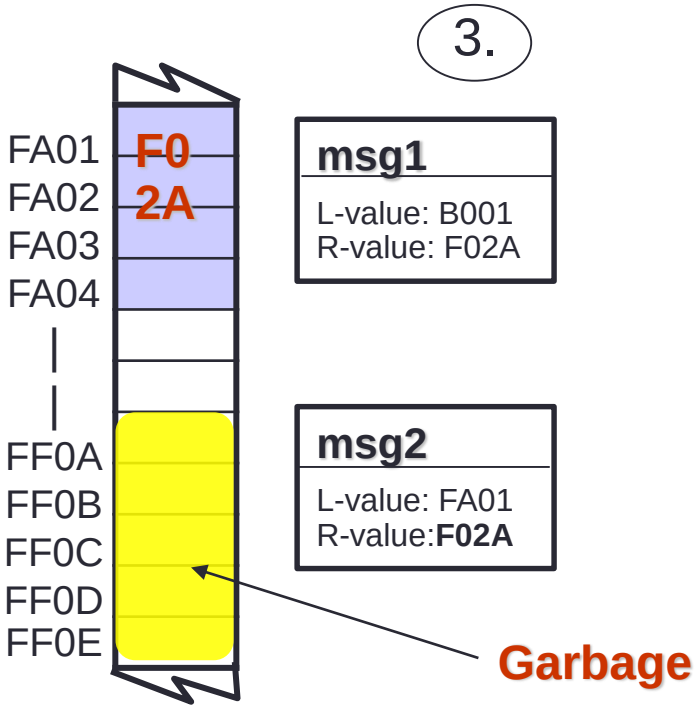
Reference Assignment

1. String msg1=new String(“str1”);
2. String msg2=new String(“str2”);
3. String msg2=msg1; (R-value Copy)



Object without Reference

- We can use an object only if we have a reference to it
- The object without reference is called “Garbage”



METHOD

```
public class BankAccount{
 private int accId;
 private double balance;
```

Attributes

```
 public void setAccId(int id){
        accId=id;
 }
 public void setBalance(double bal){
        balance=bal;
 }
 public int getAccId(){
        return accId;
 }
 public double getBalance(){
        return balance;
 }
 public void deposit(double amount){
        balance=balance+amount;
 }
 public void withdraw(double amount){
        if(amount<=balance)
           balance=balance-amount;
        else
           System.out.println("cannot withdraw please check your balance")
 }
 public String toString(){
        return "Account Id = "+ accId +"\nBalance = " +balance;
  }
}
```

Methods

Method Header

- A method declaration begins with a *method header*

```
public char calc (int num1, int num2, String message)
```

visibility modifier

return type

method name

parameter list

The parameter list specifies the type and name of each parameter

The name of a parameter in the method declaration is called a *formal parameter*

Method Body

- The method header is followed by the method body

```
public char calc (int num1, int num2, String message)
        {
            int sum = num1 + num2;
            char result = message.charAt (sum);

            return result;
        }
```

The return expression must be consistent with the return type

`sum` **and** `result` **are local data**

They are created each time the method is called, and are destroyed when it finishes executing

Local Data

- As we've seen, local variables can be declared inside a method
- The formal parameters of a method create *automatic local variables* when the method is invoked
- When the method finishes, all local variables are destroyed (including the formal parameters)
- Keep in mind that instance variables, declared at the class level, exists as long as the object exists

The return Statement

- The return type of a method indicates the type of value that the method sends back to the calling location
- A method that does not return a value has a **void** return type
- A return statement specifies the value that will be returned

```
return expression;
```

- Its expression must conform to the return type

Parameters

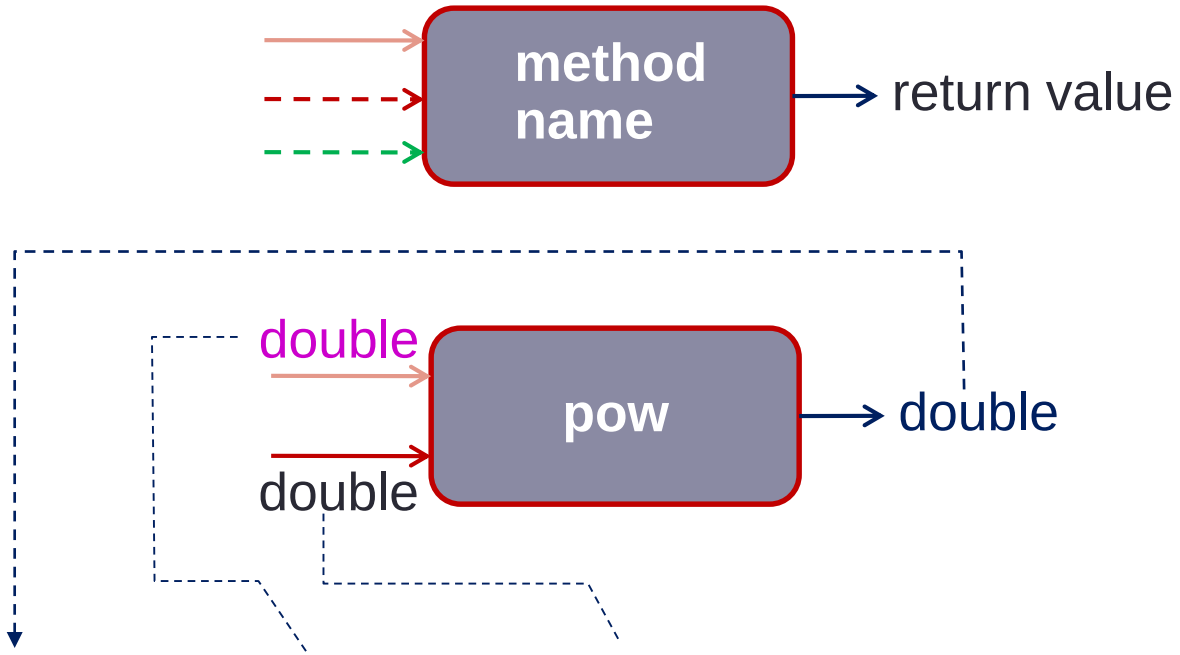
- When a method is called, the *actual parameters* in the invocation are copied into the *formal parameters* in the method header

```
ch = obj.calc (25, count, "Hello");
```

```
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```

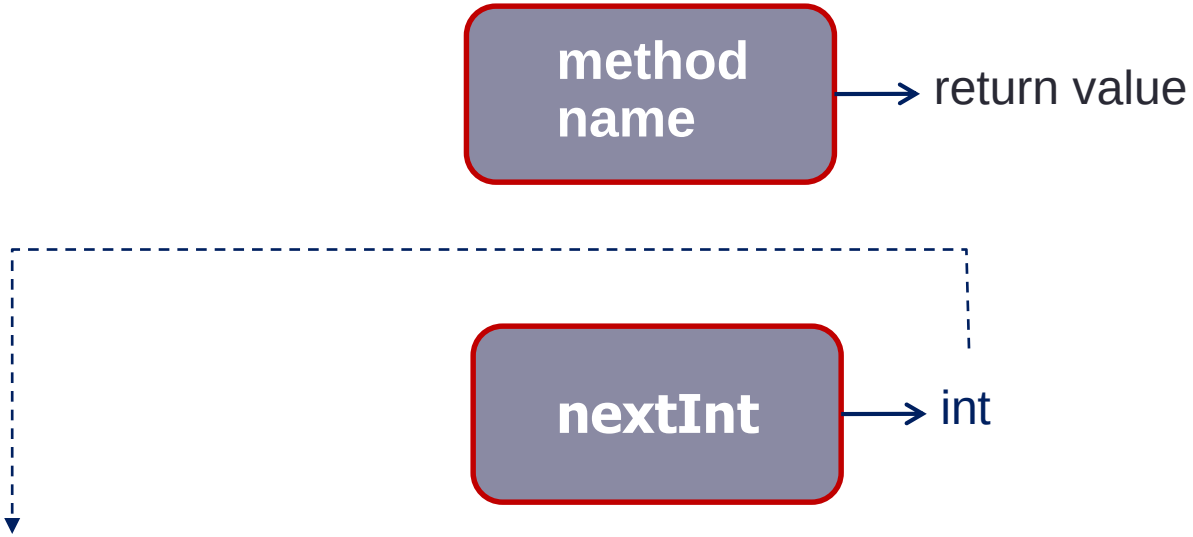
Method patterns (1):



static double pow (double a, double b)

Returns the value of the first argument raised to the second argument.

Method patterns (2):



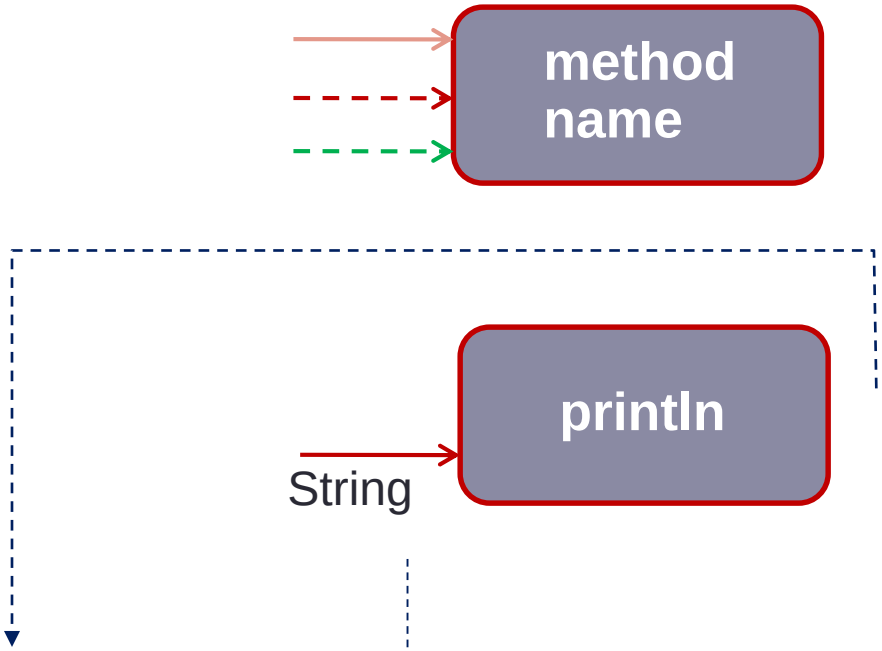
public int nextInt()

Scans the next token of the input as an int.

public static double random()

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

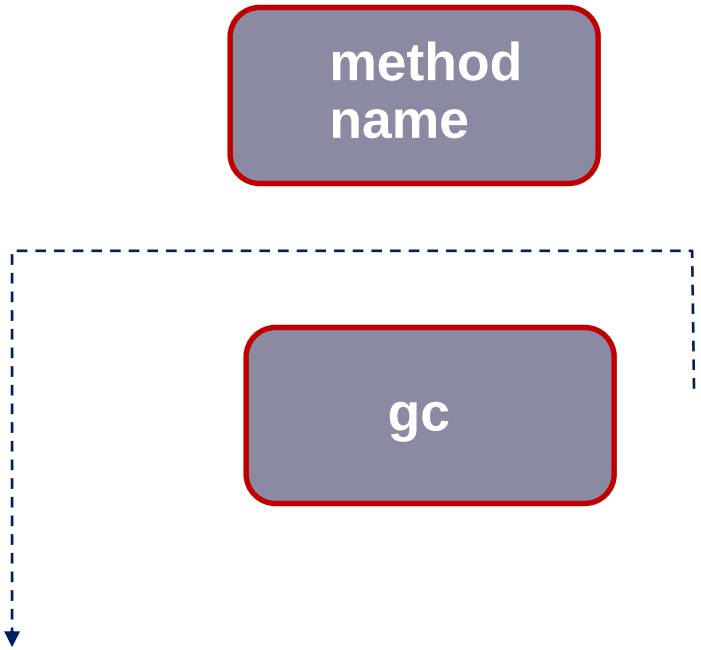
Method patterns (3):



public void println (String x)

Prints a String and then terminate the line. This method behaves as though it invokes print(String) and then println().

Method patterns (4):



public static void gc ()

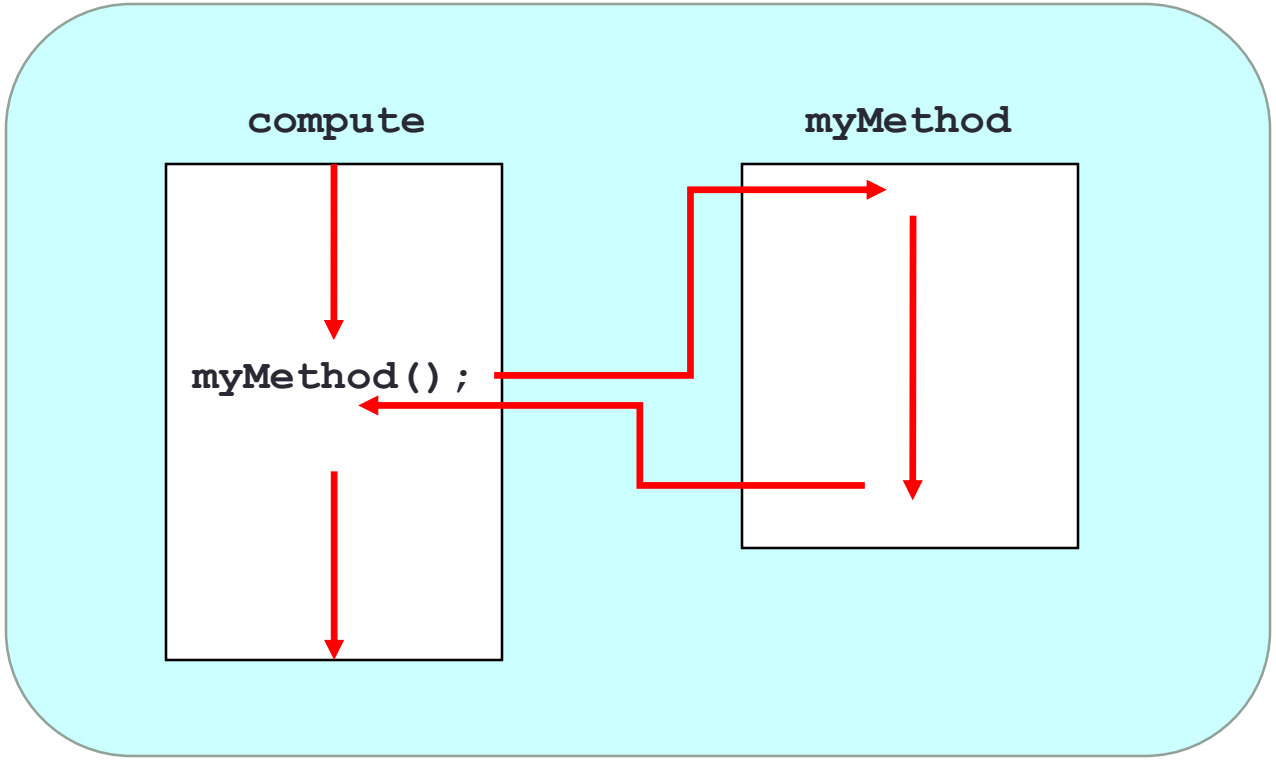
Runs the garbage collector.

Method Control Flow

- When a method is invoked, the flow of control jumps to the method and executes its code
- When complete, the flow returns to the place where the method was called and continues
- The invocation may or may not return a value, depending on how the method is defined

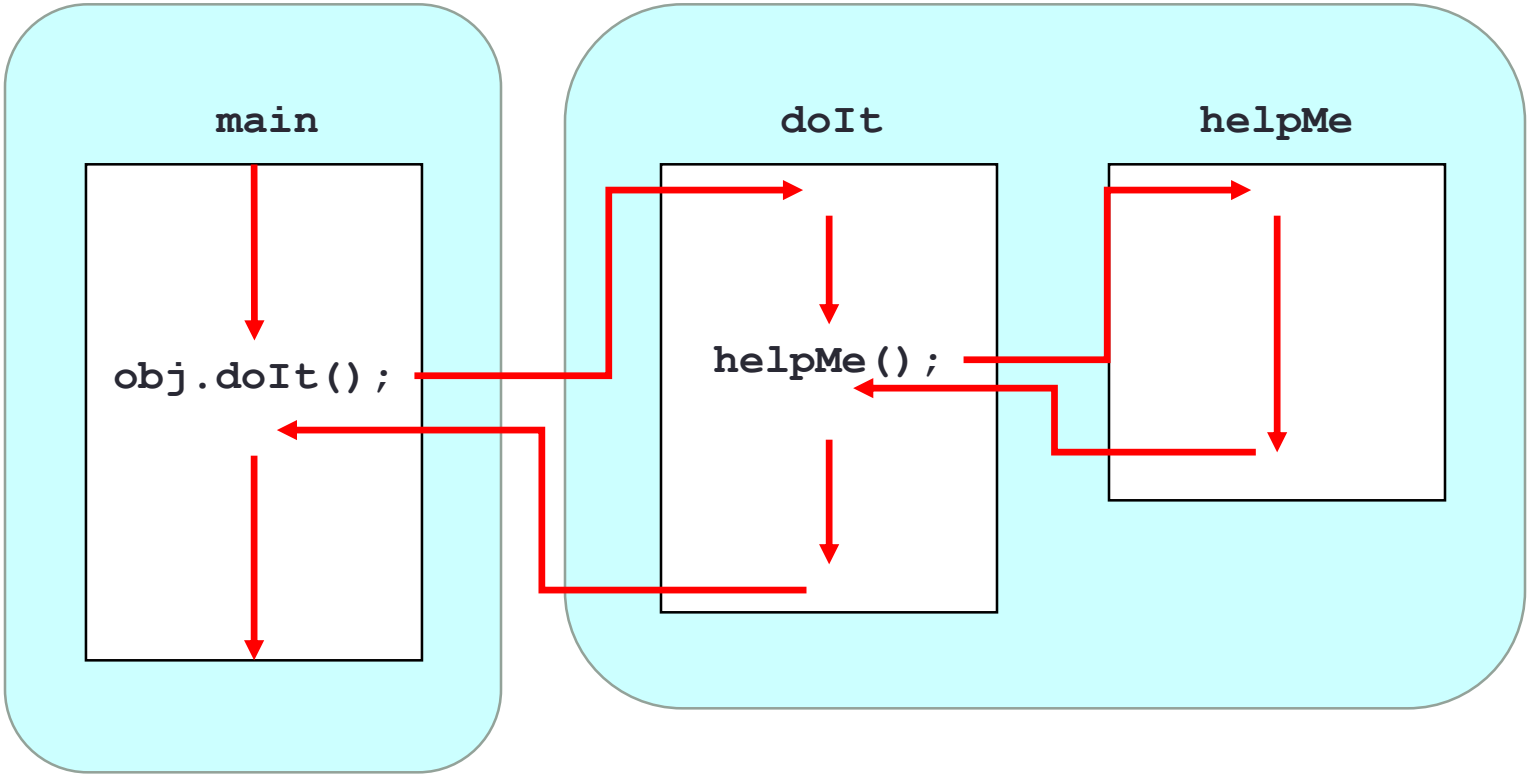
Method Control Flow

- If the called method is in the same class, only the method name is needed



Method Control Flow

- The called method is often part of another class or object



Visibility Modifier

Modifier	Explanation
(default)	A class, constructor, method, or data field is visible in this package
public	A class, constructor, methods, or data field is visible to all the programs in any package
private	A constructor, method, or data fields is only visible in this class
protected	A constructor, method, or data field is visible in this package and in subclasses of this class in any package

Visibility Modifiers

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**

Static Class Members

- Recall that a static method is one that can be invoked through its class name
- For example, the methods of the `Math` class are static:

```
result = Math.sqrt(25)
```

- Variables can be static as well
- Determining if a method or variable should be static is an important design decision

The static Modifier

- We declare static methods and variables using the `static` modifier
- It associates the method or variable with the class rather than with an object of that class
- Static methods are sometimes called *class methods* and static variables are sometimes called *class variables*
- Let's carefully consider the implications of each

Static Variables

- Normally, each object has its own data space, but if a variable is declared as static, only one copy of the variable exists

```
private static float price;
```

- Memory space for a static variable is created when the class is first referenced
- All objects instantiated from the class share its static variables
- Changing the value of a static variable in one object changes it for all others

Static Methods

- The methods that a class definition has are called static methods.
- A static method is a characteristic of a class, not of the objects it has created.
- Important:
  - A program can execute a static method without first creating an object!
  - All other methods (those that are not static) must be part of an object. An object must exist before they can be executed.

Static Methods

```
class Helper{
   public static int cube (int num){
      return num * num * num;
   }
}
```

Because it is declared as static, the method can be invoked as

```
value = Helper.cube(5);
```

Static Class Members

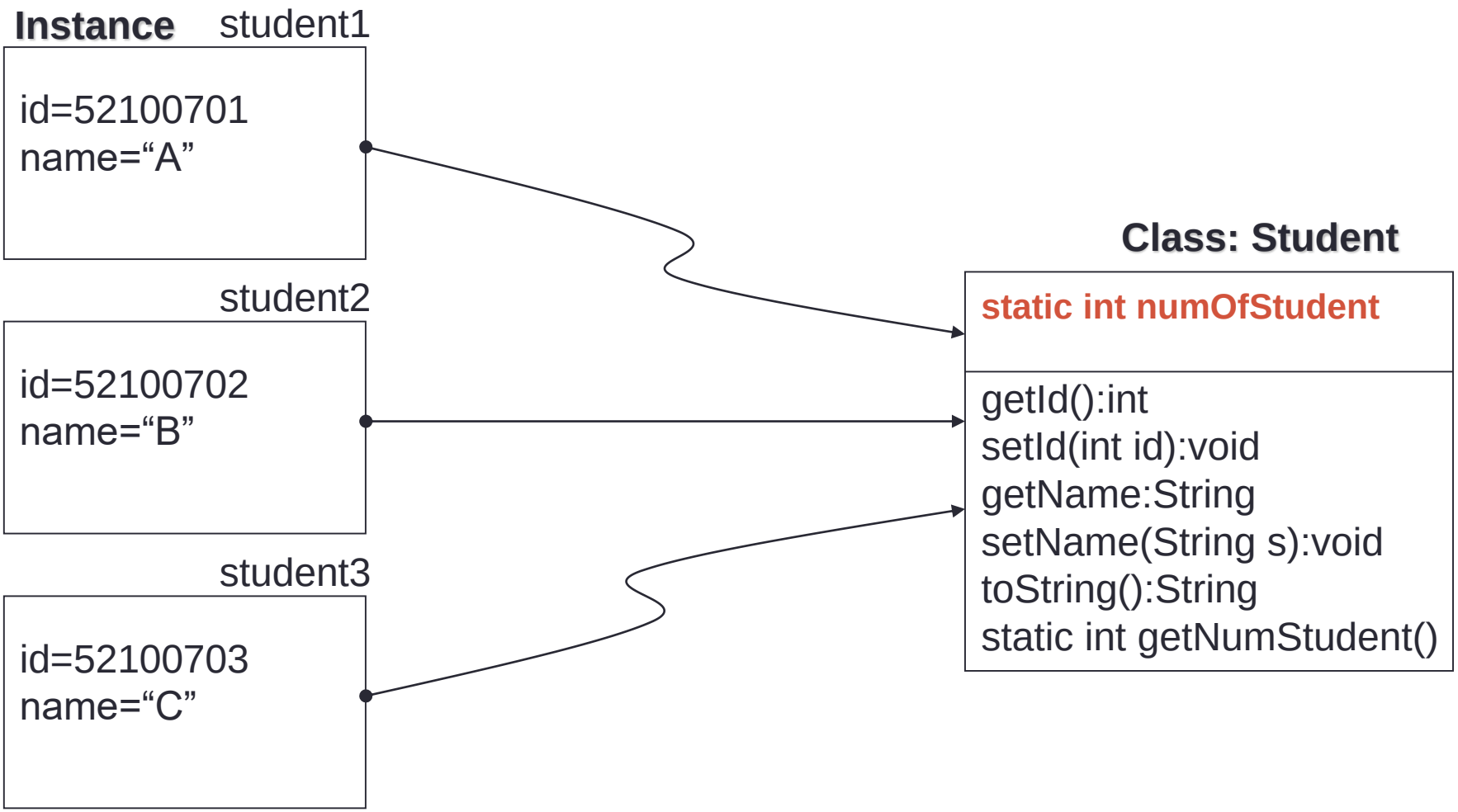
- The order of the modifiers can be interchanged, but by convention visibility modifiers come first
- Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object
- Static methods **cannot reference instance variables** because instance variables don't exist until an object exists
- However, a static method **can reference static variables or local variables**

Student Class Example

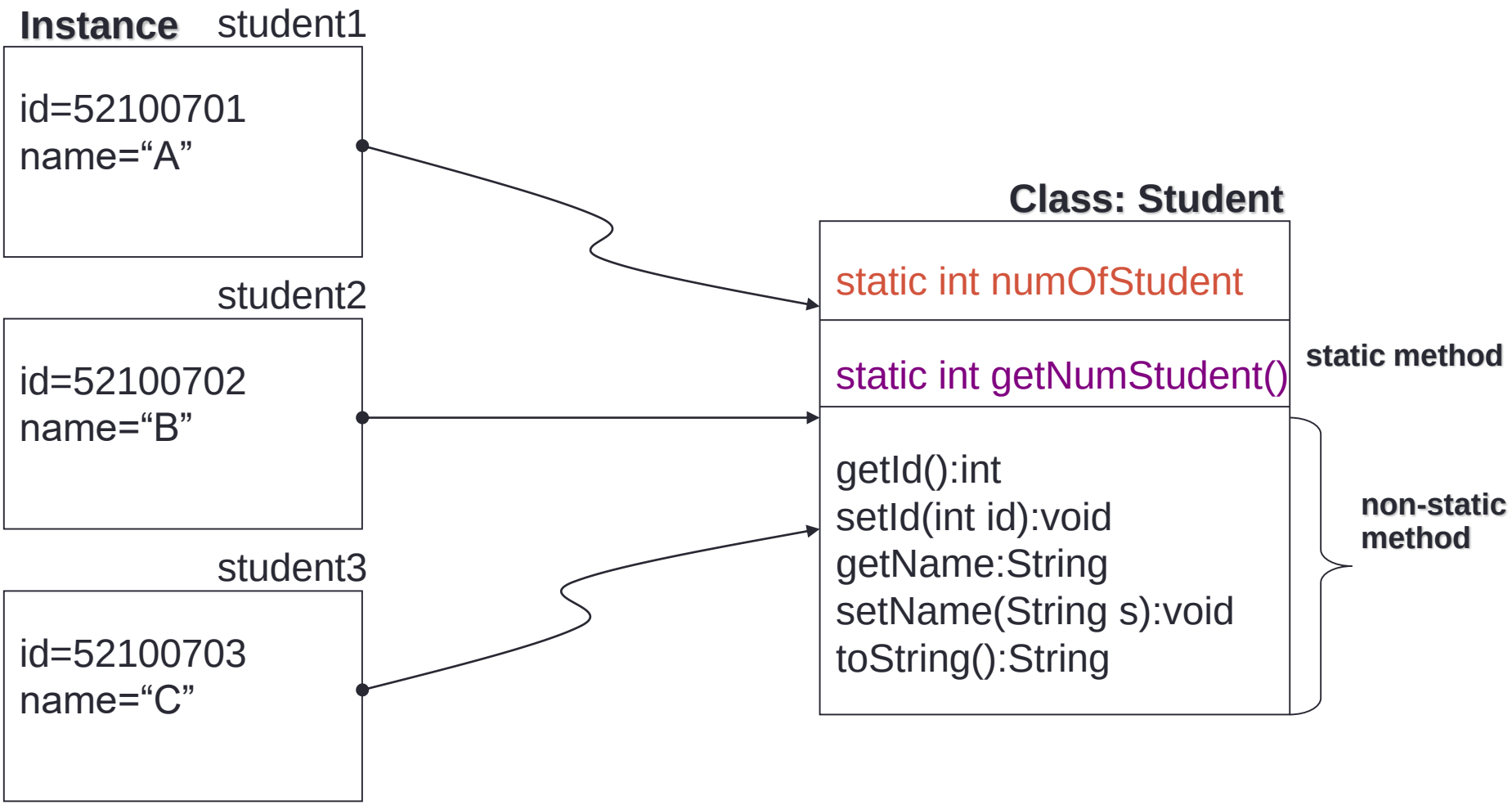
Class: Student

int id String name static int numOfStudent
getId():int setId(int id):void getName:String setName(String s):void toString():String static int getNumStudent()

Class Variables Vs. Instance Variables



Class Methods Vs. Non-Static Methods



Static Class Members

- Static methods and static variables often work together
- The following example keeps track of how many `Slogan` objects have been created using a static variable, and makes that information available using a static method

Slogan.java

```
public class Slogan{
        private String phrase;
        private static int count = 0;

//  Returns this slogan as a string.
        public String toString()
        {
                return phrase;
        }

//  Returns the number of instances of this class that have been created.
        static public int getCount()
        {
                return count;
        }
        public void setPhrase(String s)
        {
                phrase=s;
                count++;
        }
}
```

SloganCounter.java

```
public class SloganCounter{
//  Creates several Slogan objects and prints the number of
//  objects that were created.
        public static void main (String[] args){
                Slogan obj1;
                obj1 = new Slogan ();
                obj1.setPhrase("Remember the Alamo.");
                System.out.println (obj1);
                Slogan obj2 = new Slogan ();
                obj2.setPhrase("Don't Worry. Be Happy.");
                System.out.println (obj2);
                Slogan obj3 = new Slogan ();
                obj3.setPhrase("Live Free or Die.");
                System.out.println (obj3);
                Slogan obj4 = new Slogan ();
                obj4.setPhrase("Talk is Cheap.");
                System.out.println (obj4);
                Slogan obj5 = new Slogan ();
                obj5.setPhrase("Write Once, Run Anywhere.");
                System.out.println (obj5);
                System.out.println();
                System.out.println ("Slogans created: " + Slogan.getCount());
                System.out.println("Slogans created: " +obj1.getCount());
                System.out.println("Slogans created: " +obj2.getCount());
                System.out.println("Slogans created: " +obj3.getCount());
                System.out.println("Slogans created: " +obj4.getCount());
                System.out.println("Slogans created: " +obj5.getCount());
              }
}
```