Computers have RAM and disk memory

Answer: Magic.

To this day, every time I write a program that successfully compiles and runs, I find magical.

In the beginning: I was mystified by the magic.

Now: I'm a magician.

You all: Will be magicians too!

Here's what's going on ...

What actually happens when I write compile and run a program?

Answer: Magic.

To this day, every time I write a program that successfully compiles and runs, I find magical.

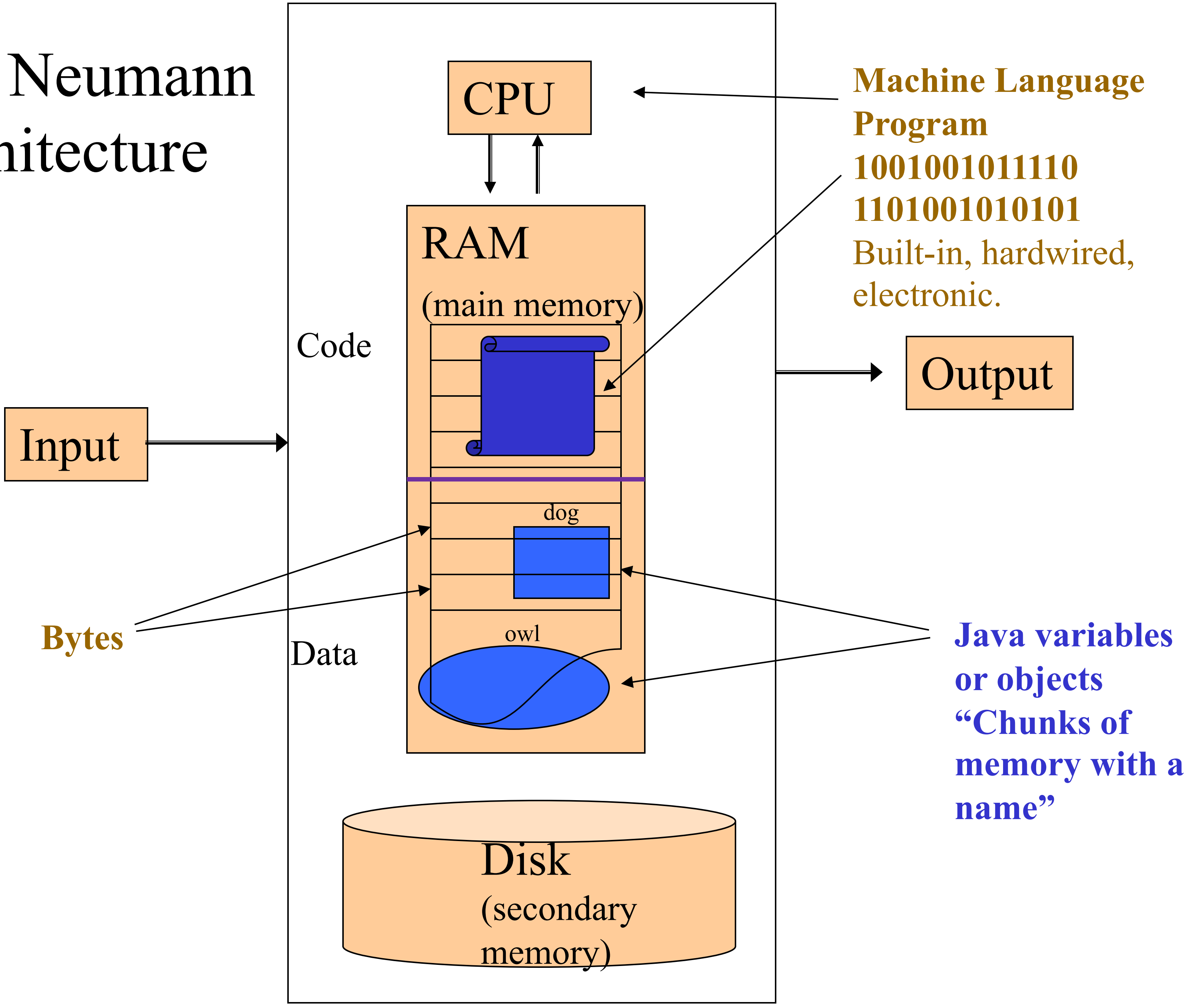
In the beginning: I was mystified by the magic.

Now: I'm a magician.

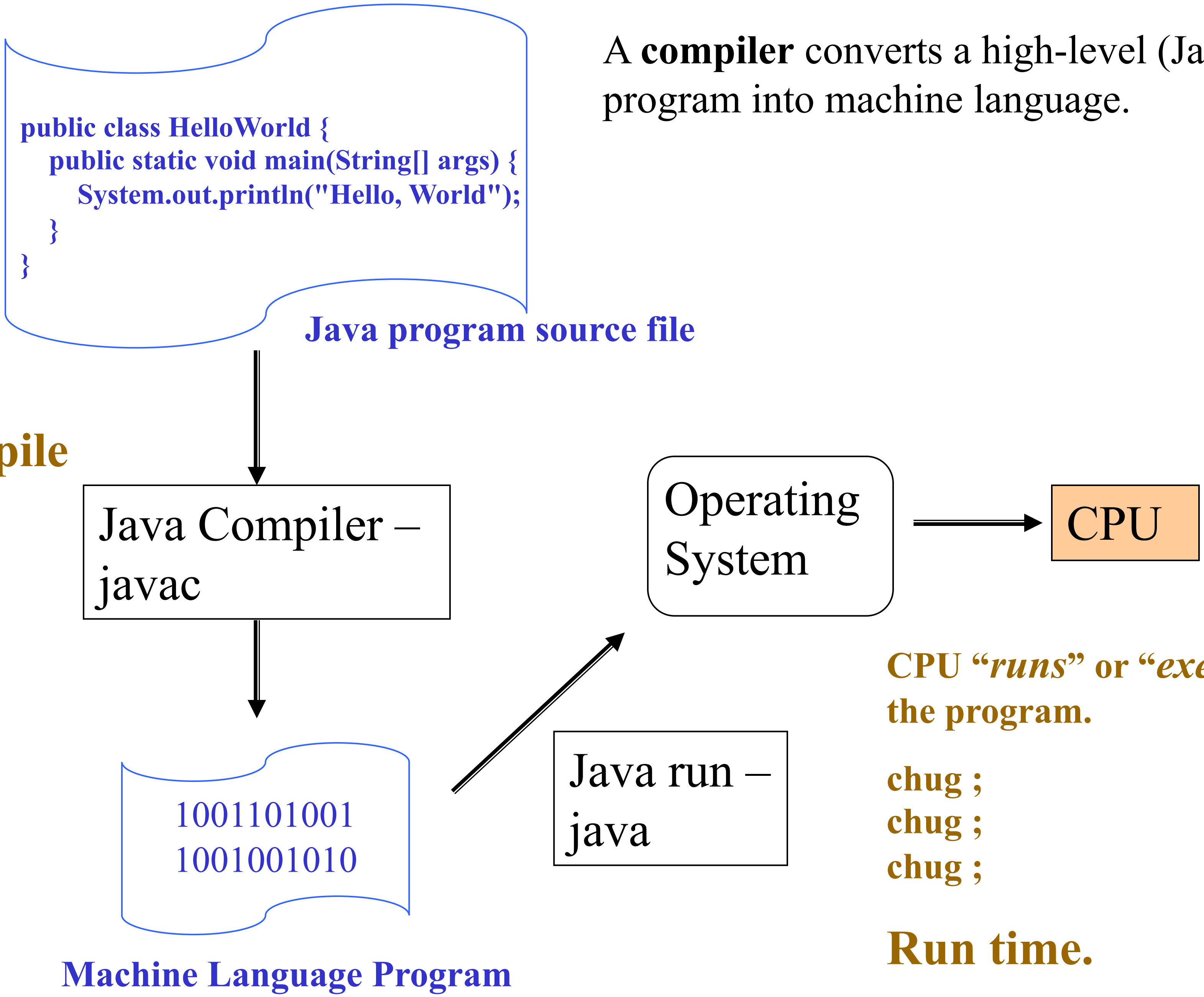
You all: Are apprentices and need this course and one other to become a magician.

Here's what's going on ...

Von Neumann Architecture



A **compiler** converts a high-level (Java) program into machine language.



```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

Compile time.

CPU “*runs*” or “*executes*” the program.

chug ;
chug ;
chug ;

Run time.

An object is a chunk of memory (RAM) that exists at runtime

```
class Main {
 String fName = new String("William");
 String lName = new String("Sakas");
 String mName = new String("Gregory");
}
```

This code creates three objects, three chunks of memory, **structured** in such a way to hold a string.

Java provides the **structure** for Strings, soon we'll see how the human programmer provides the **structure** for a Money object using Java **classes**.

So, for us: An object == A chunk of memory

Hence a String that exists in memory at runtime is an object.

Disclaimer: Other built-in types, like ints, which are also chunks of memory that exist at runtime are not consider objects in Java, but rather declarations of "primitive types."

Java forces this distinction, by requiring the programmer to capitalize data types that create objects, vs. data types that declare primitive chunks of memory.

int x;

String aString;

I find this more distracting than useful (at least IMHO) and will often use "object" to mean any declared variable that is will be created at runtime.

In object-oriented languages like Java, objects consist of *two* parts:

- the ***structure*** of data (e.g., Strings are containers of characters.), and
- the ***functionality*** of data (e.g. .length(), .charAt( ), etc)

For example, consider the object *declaration*:

```
String lName  = "Sakas" ;
```

creates a chunk of memory, **structured** in such a way to hold the characters: *Sakas*.

lName is a String, and as we know, String objects come with with some **functionality**:

```
lName.charAt(3)  ;
```

read "in position access " **functionality**

```
if (lName.equals("Smith") )
```

comparison **functionality**

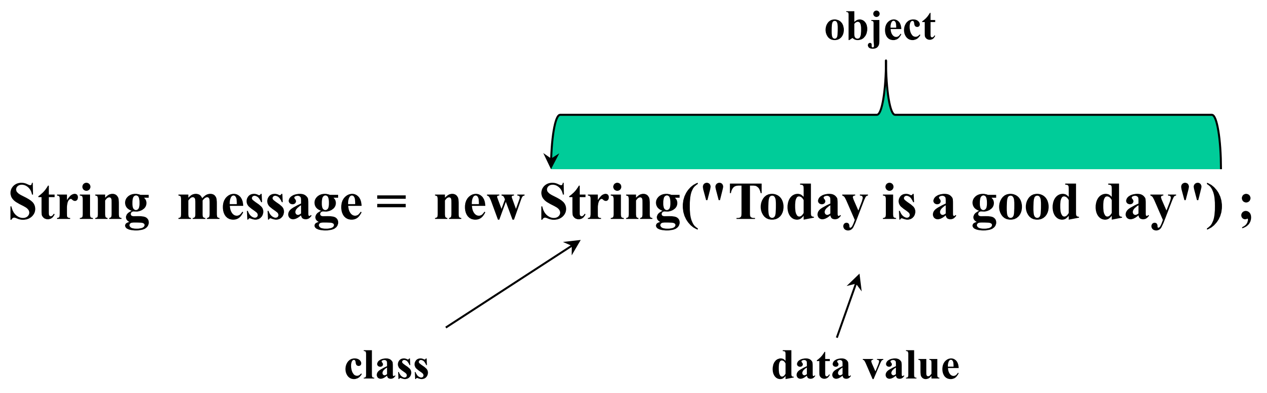
.

.

.

We say String is a ***class*** **because** string objects have both:

structure **and** ***functionality***.



```
String  message =  new String("Today is a good day") ;
```

Since classes provide both structure and functionality, it is good to think of an object declaration in Java as creating **two** places in main memory (RAM) for each object:

One place where the data values are **"structured"** and stored and

another place where **"functionality"** executes.

Technically (most) all modern computer languages work like this —

Von Neumann Architecture.

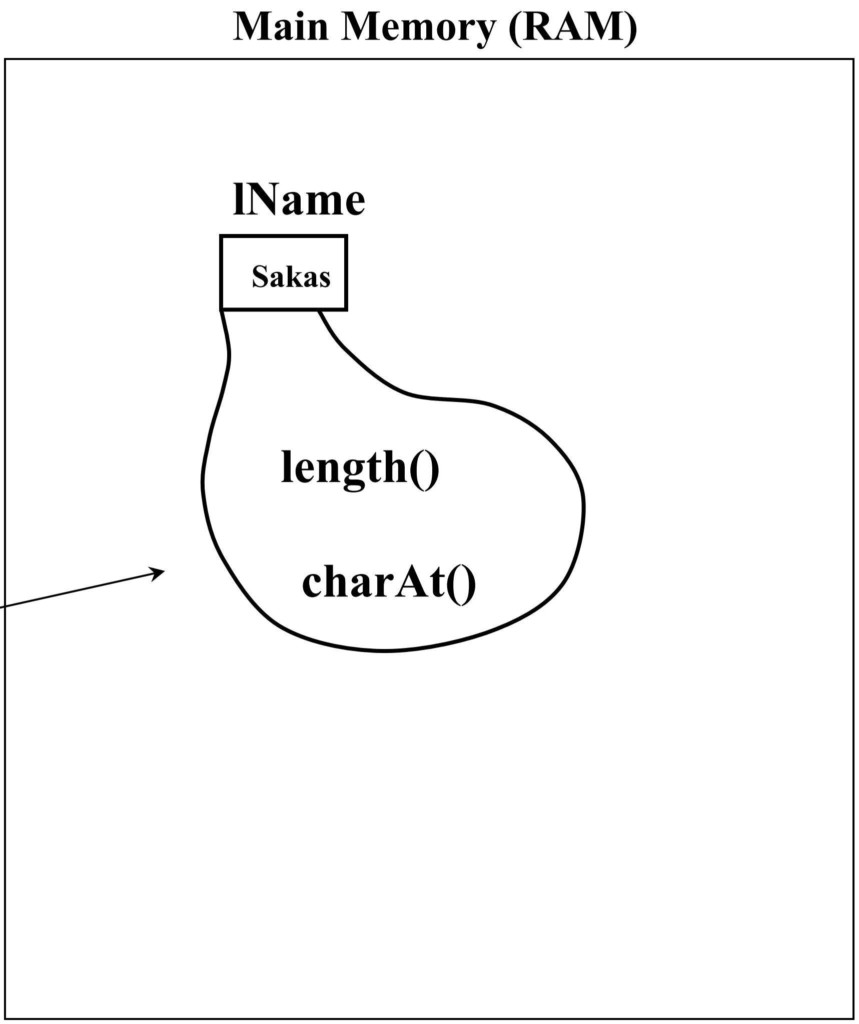
An example

A String object in memory

lName = new String("Sakas");

char ch = lName.atChar(2);

int size – lName.length();



Method Space
for an object
This is where the "functionality" executes. For example, charAt(), length(), etc.

"Elbow room" for functionality - always a good thing!

When an object is declared, an **"method space"** for the object is allocated in main memory. Functions and operators that are associated with the object execute in the object's method space.*

* "***method space***" is my term, you won't find it online or in a textbook.

This is an abstract and informative way of looking at objects in Java. However, it is important to realize that **exactly** where in memory an operator executes is up to the compiler implementation and depends on the OS and computer hardware.

Now some Java syntax

Review. String *length* function

```
String dog = "hello";

dog . length() ;
```

$\Downarrow$

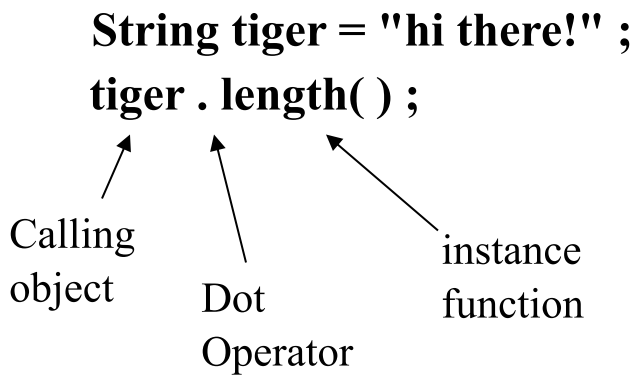
5

```
String tiger = "hi there!" ;

tiger . length () ;
```

$\Downarrow$

9



```
String tiger = "hi there!" ;
tiger . length( ) ;
```

dot operator:

Execute the **instance method** on the right hand side (RHS) (in this example is *length ( )*) in the ***mehtod space*** of the **calling object** on the LHS (calling object here is *tiger*)

ALWAYS: **OBJECT - DOT - instance** **repeat it to yourself!**

(not: class-dot-instance or function-dot-instance or anything else!)

Note: Technically in Java *tiger* is a reference to the object in memory, not the object itself. This may become important in your data structures course, but we can just call the variable name "an object."

It's easy to mix up class definitions and objects.

The important thing is that:

- **classes** are **definitions** or **blueprints.**
  Classes don't exist in memory, they only define what CAN exist in memory.

- **objects** physically exist in main memory when they are **declared** and a program is run.

A class* defines** the **shape and function** of an ***object

shape	= =	chunk of memory that is allocated and how it is organized
		-- instance variables
function	= =	instance functions and operators (e.g., length(), etc.)

Class objects are chunks of memory that have structure and functionality.

These chunks can be complicated and can contain several (or many) **attributes / data instances / fields / slots** that describe the object in the "real world."

Also, **instance functions,** in Java often **methods**) can be attached to objects in order to manipulate the **attributes**.

It is possible for a programmer to define his or her **own** classes (or data types).

This is useful when **more than one value** describes an **object** in the real world.

Examples of useful classes:

student

professor

money

time

day of the year (Feb 9)

duration

Let's design a program that can deal with students.

First question, (as a program philosopher or **design analyst**) is:

What characteristics or attributes represent a student in the "real world?"

1) ID number

2) gender

3) first name and last name

4) date started

5) balance due

6) amount paid

7) address

8) phone number

9) age

10) ft/pt

11) financial aid info

.

.

.

Simplified Student as an example with just three attributes and 2 methods:

DATA INSTANCES:

1) ID number - string data type

2) FtPt - char data type

3) balance due - double data type

METHODS or FUNCTION INSTANCES:

1) input

2) output

Now that the program philosopher or analyst is done, our goal as programmers is to be able to **create student class objects** that:

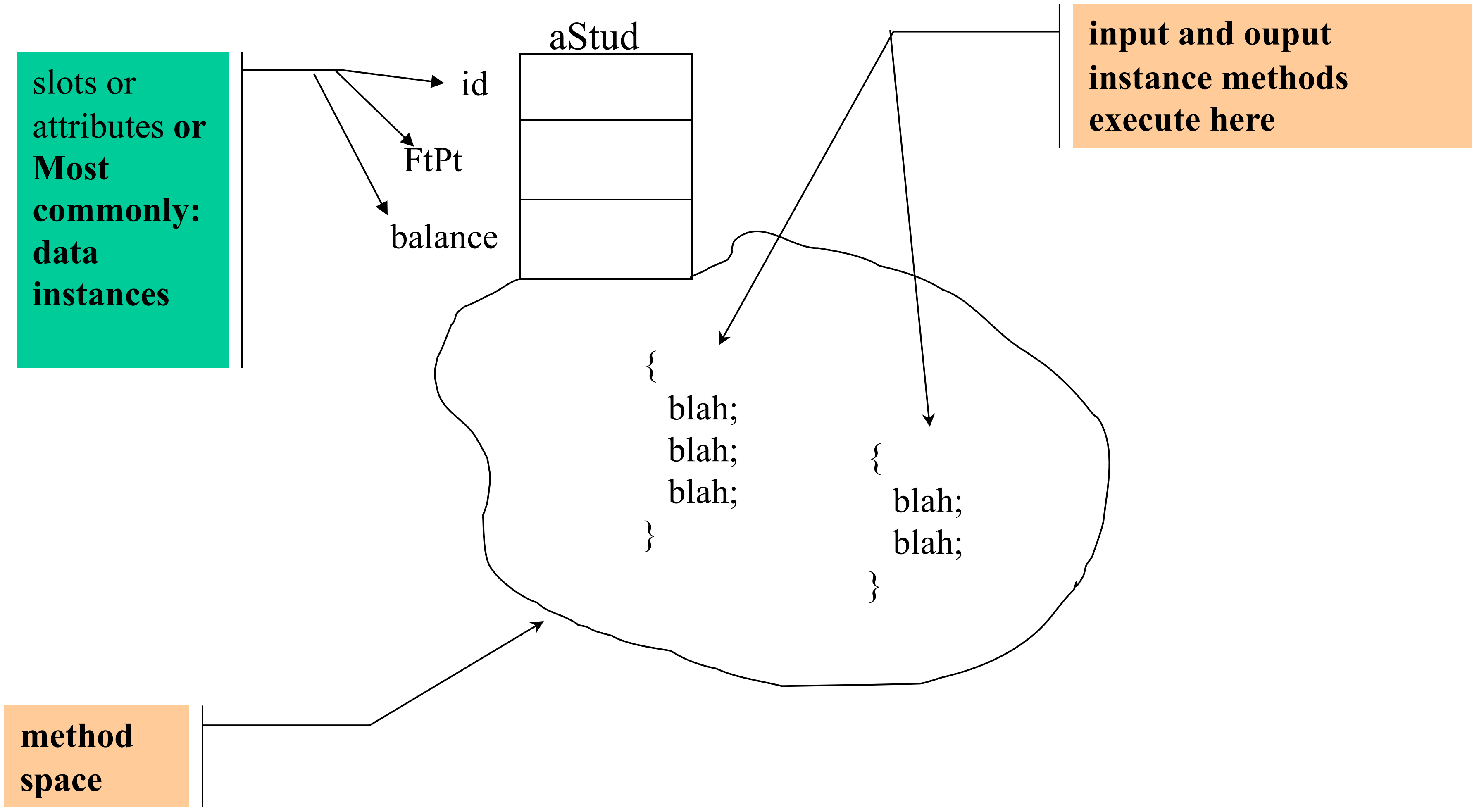
1) that contain three data values and have

2) input and output functionality.

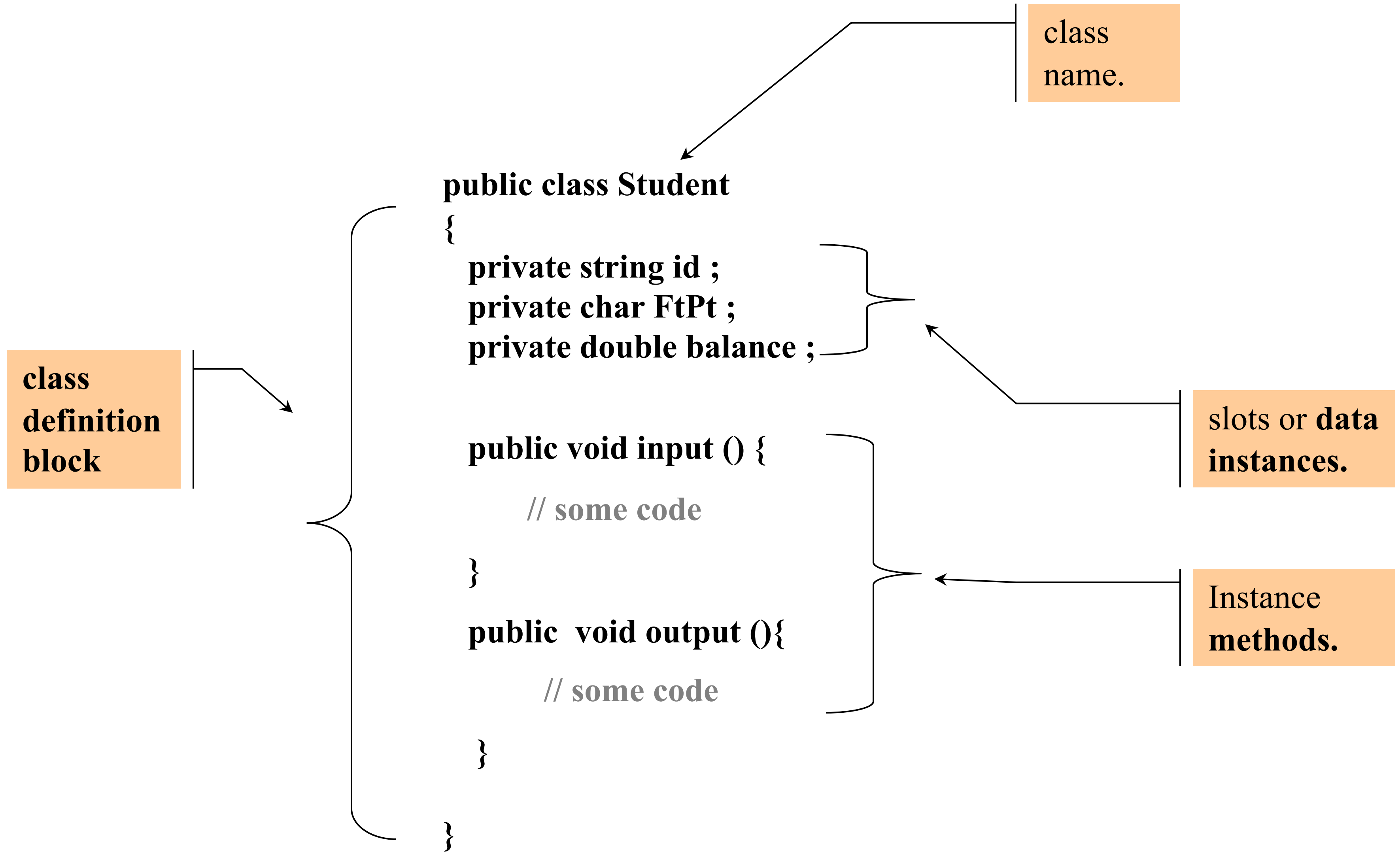
Basically, we want to declare complicated objects simply, like this:

```
Student  aStud ;            // review: What is Student? class, object or function?
```

Student aStud ; // creates a student ***object*** in RAM


slots or attributes **or Most commonly: data instances**
id
FtPt
balance
aStud
input and ouput instance methods execute here
{ blah; blah; blah; }
{ blah; blah; }
method space

A *class* is a definition or a description or a **blueprint** of what ALL objects of the class will look like



```
public class Student
{
   private string id ;
   private char FtPt ;
   private double balance ;

   public void input () {

         // some code

   }
   public  void output (){

          // some code

    }

}
```

Let’s design a program that can deal with students. Here’s a slightly different definition (so that words and code can fit on the slides):

Simplified student for our next example:

DATA instances:

1) name - string data type

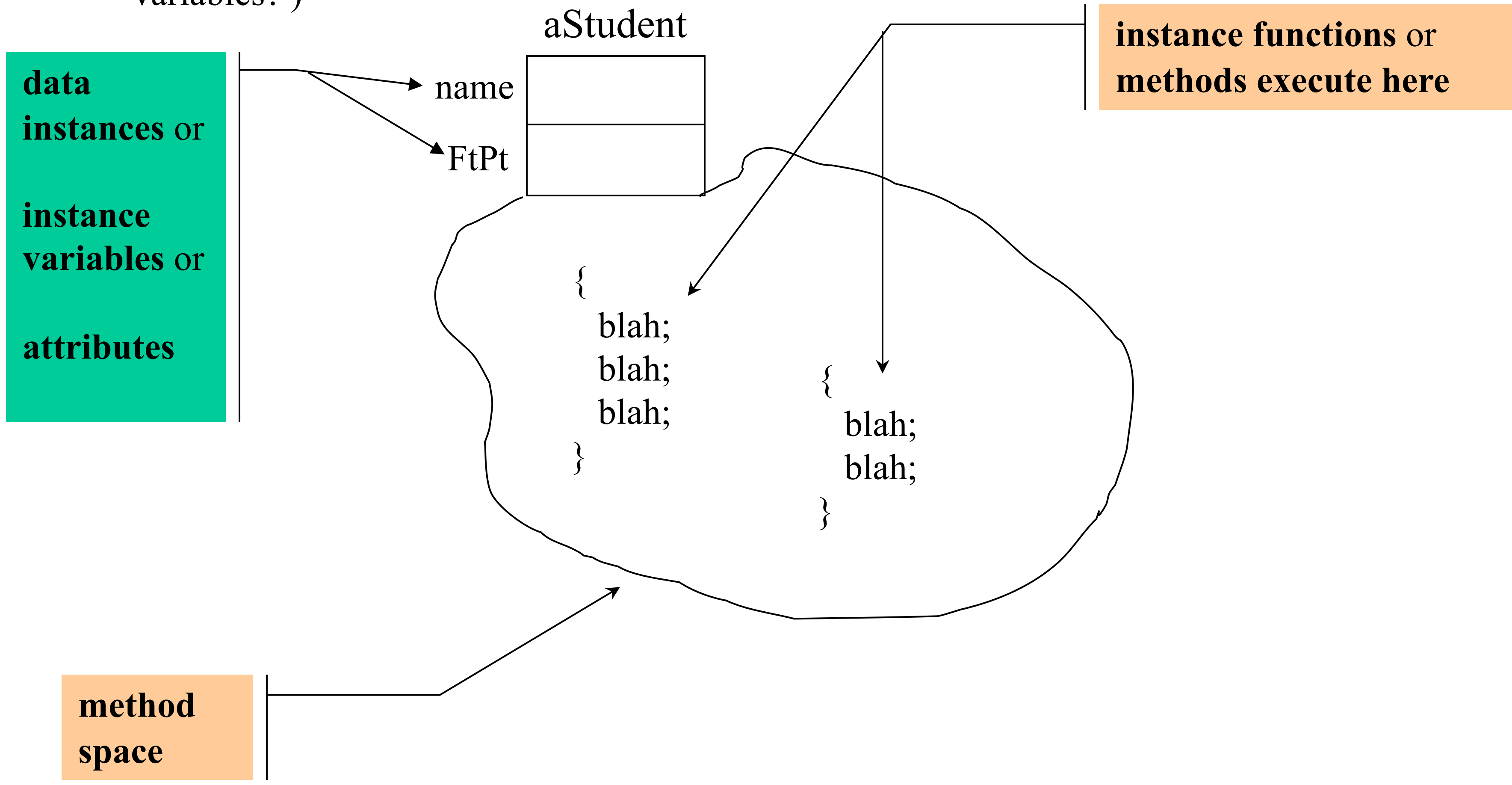
2) FtPt - char data type

METHODS or instance FUNCTIONS:

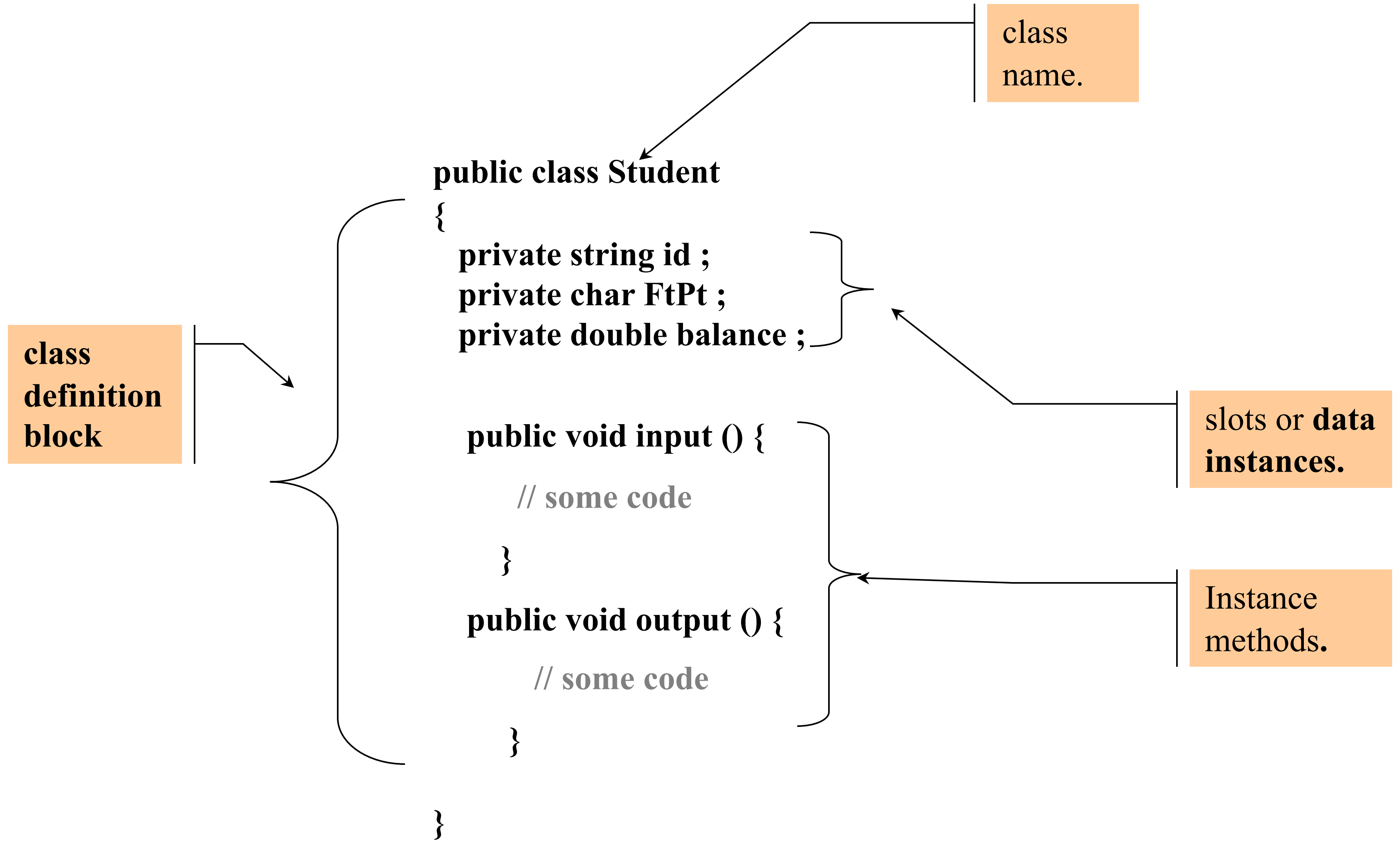
1) enterInfo

2) showInfo

aStudent == a student ***object*** in RAM (Question: How was allie declared? What is the correct syntax? What values are initially in the instance variables?)


aStudent
name
FtPt
{ blah; blah; blah; }
{ blah; blah; }
data instances or **instance variables** or **attributes**
instance functions or **methods execute here**
method space

A ***class*** is a definition or a description of what ALL objects of the class will look like



```
public class Student
{
   private string id ;
   private char FtPt ;
   private double balance ;

    public void input () {

          // some code

       }
    public void output () {

          // some code

        }

}
```

Let' s design a program that can deal with grades. In our example, grades have three basic *attributes.*

1) the name or ID of the exam the grade represents ("midterm", "final", "quiz1", etc.)
2) the grade a student attains
3) the percentage weight the grade receives in computing an average (60%, 30%, etc)

Here' s a definition or *specification* for a Grade class:

DATA instanceS:

1) ID	**- string data type** (For example: **"quiz1"**)
2) score	**- double data type**
3) weight	**- double data type**

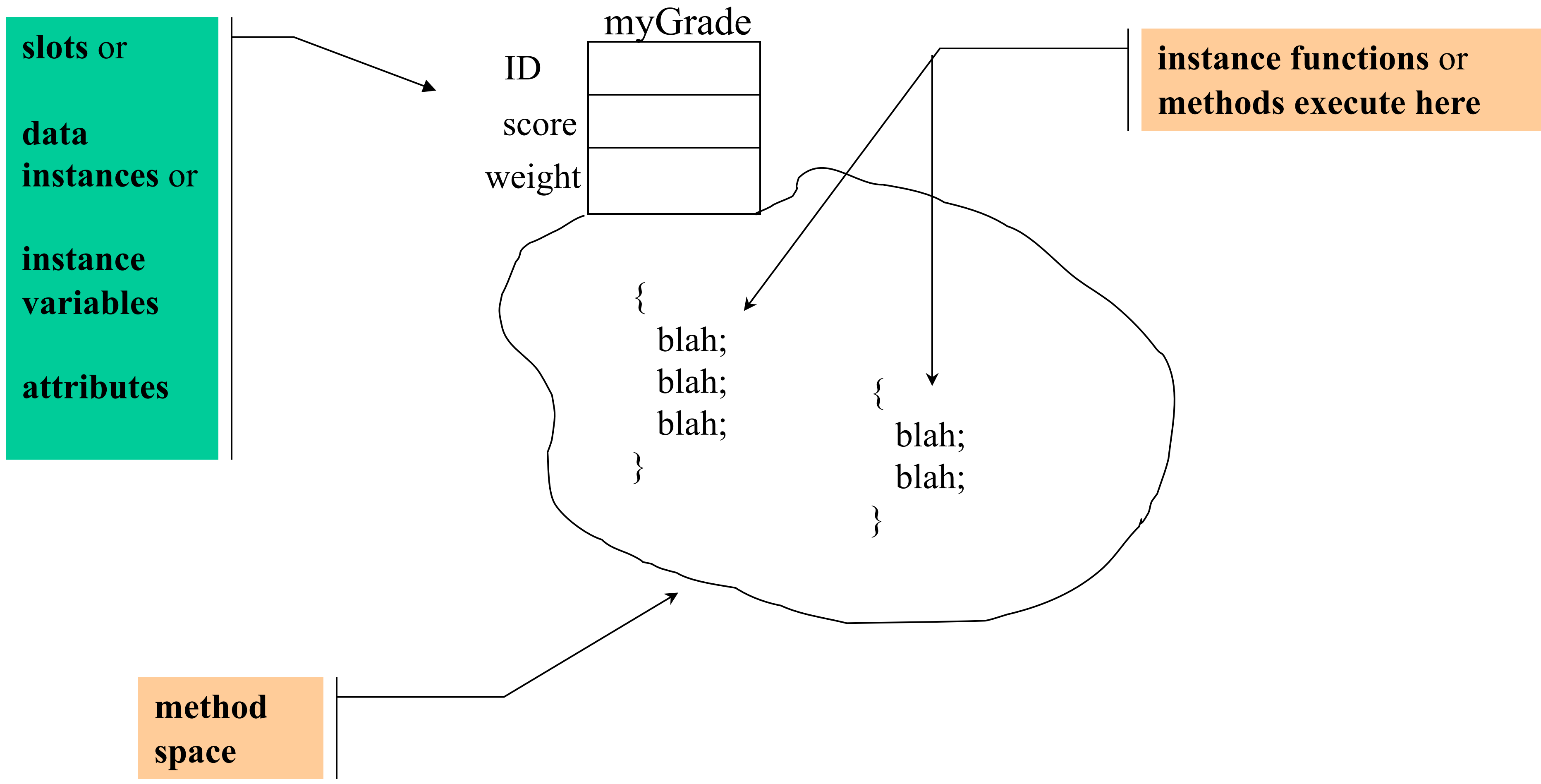
METHODS or instance FUNCTIONS:

1) enterID

2) showID

3) showScore

myGrade: a Grade ***object*** in RAM



Grade.java

```
class Grade {
 private String ID;
 private double score;
 private double weight;

 public  Grade (String id, double s, double w){
  ID = id;
  score = s;
  weight = w ;
 }
 public void enterInfo( ){
  Scanner input = new Scanner(System.in);
  ID = input.nextLine();
  score = input.nextDouble();
  input.nextLine();
  weight = input.nextDouble();
  input.nextLine();
 }
 public void showInfo( ){
  System.out.println("ID: "+ ID +
                  " Score: " + score +
                  " Weight: " + weight);
 }
}
```

Main.java

```
class Main {

public static void main(String[] args) {
   Grade g = new Grade("3456",95,0.10);
   g.showInfo( );
   g.enterInfo( );
   g.showInfo( );
  }

}
```

A big difference between data instances and objects is that:

Data instances that are defined in a class definition block
are global to *all instance functions of the class.*

It turns out that a class data instance (or "instance variable") is "**defined**" not "**declared**"

A class **definition** just says how to structure memory **when (**and **if)** an object is created but the definition **doesn't** create it.