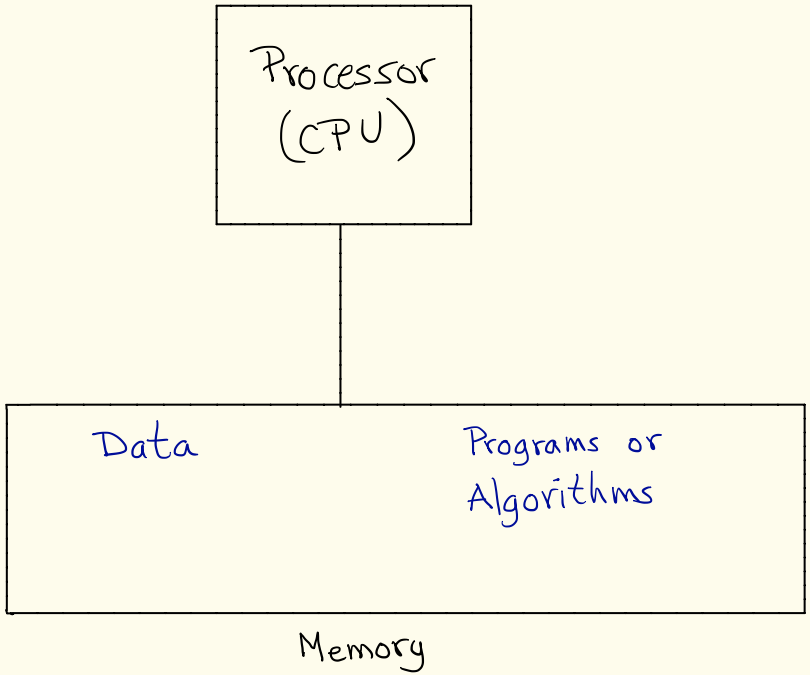
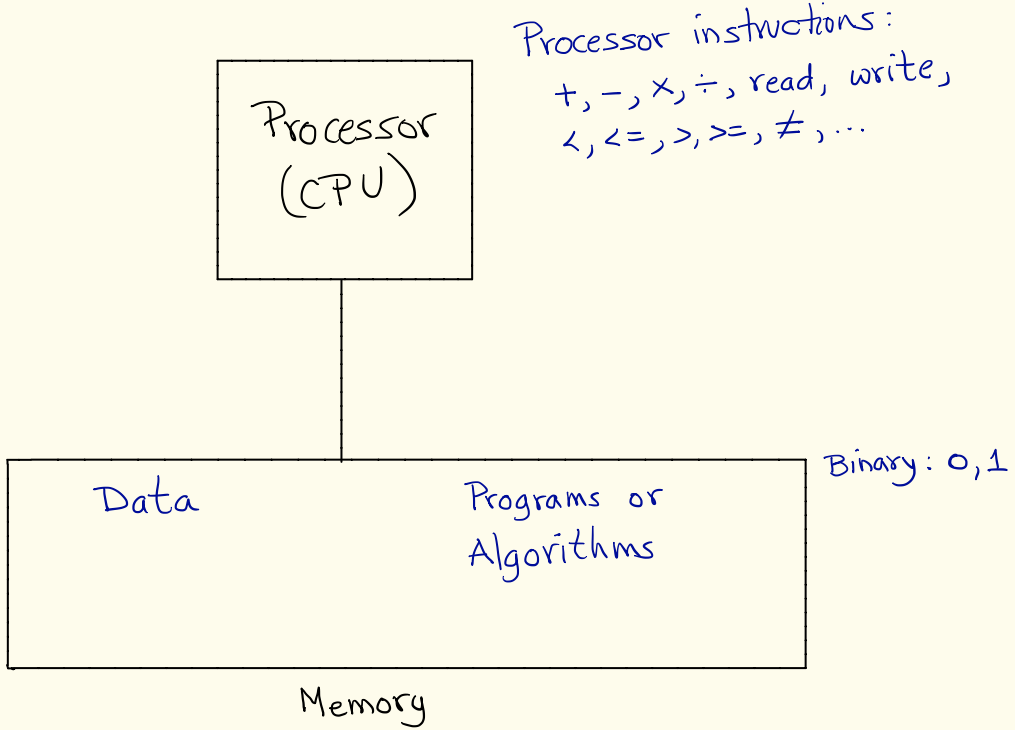
Von Neumann Model



Von Neumann Model



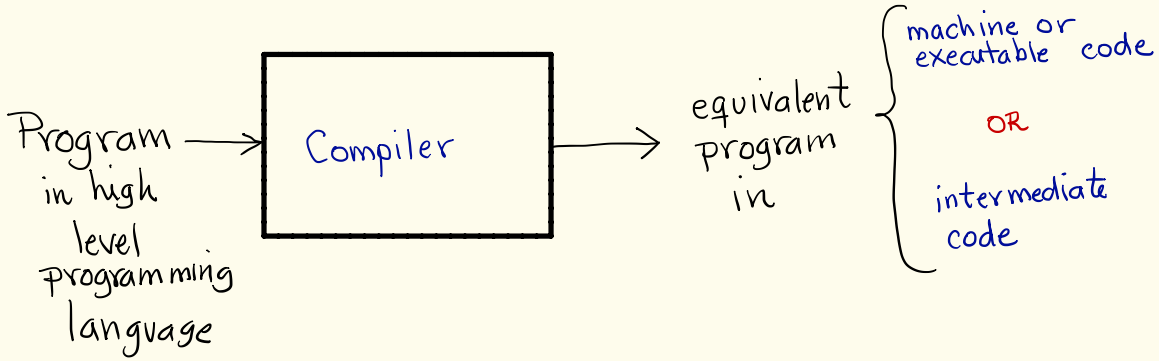
```
public class PrintHello {

        public static void main (String[] args) {
                System.out.println(“Hello”);
        }
}
```

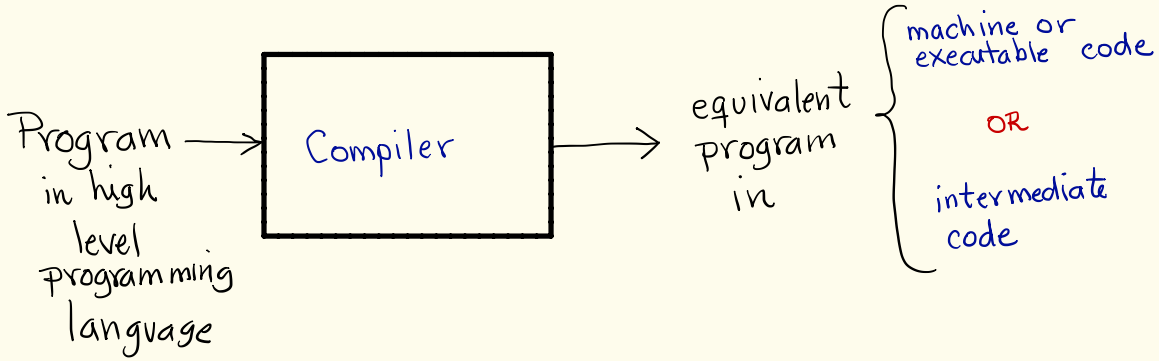
for processor 8086

Binary program that prints “hello”

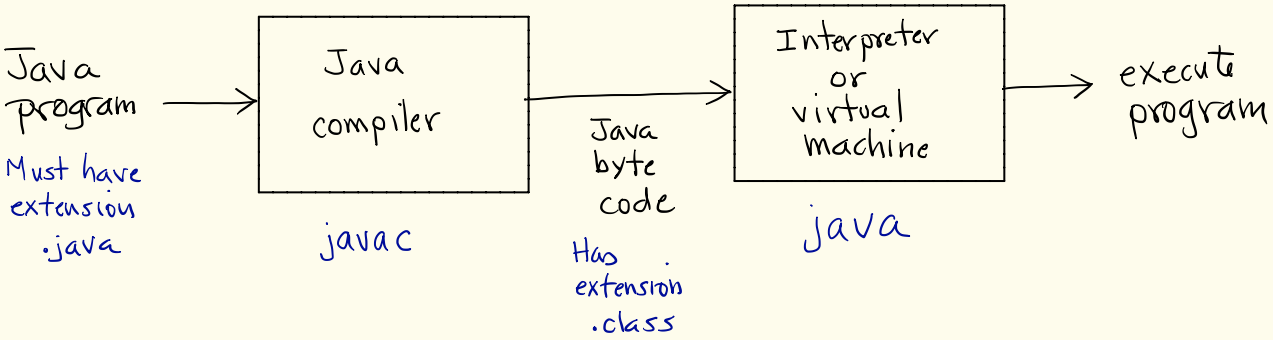
```
10111010000011000000000110110100
00001001110011010010000110111000
00000000010011001100110100100001
01001000011001010110110001101100
01101111001011000010000010101111
01101111011100100110110001101000
00100001000011010000110001101111
```

Machine or executable code


Program in high level programming language
Compiler
equivalent program in
machine or executable code
OR
intermediate code



Python and Java compilers produce intermediate code called byte code.


Java program
Must have extension .java
Java compiler
javac
Java byte code
Has extension .class
Interpreter or virtual machine
java
execute program

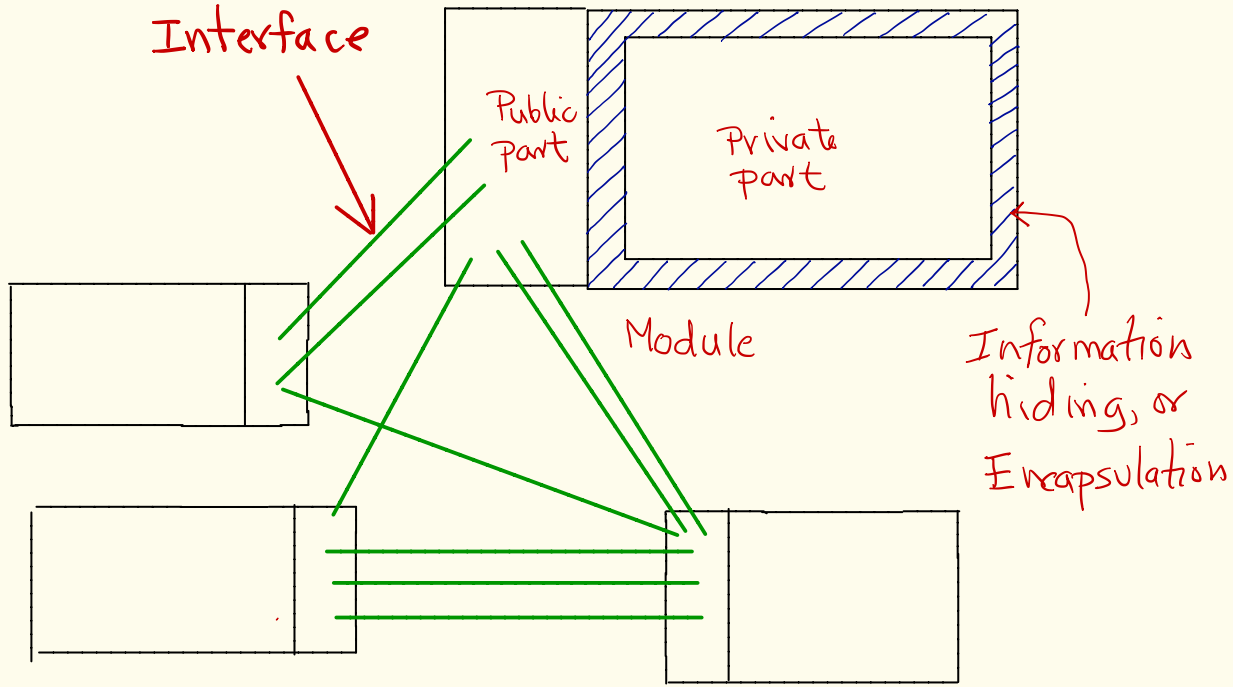
```
public class PrintHello {

        public static void main (String[] args) {
                System.out.println(“Hello”);
        }
}
```

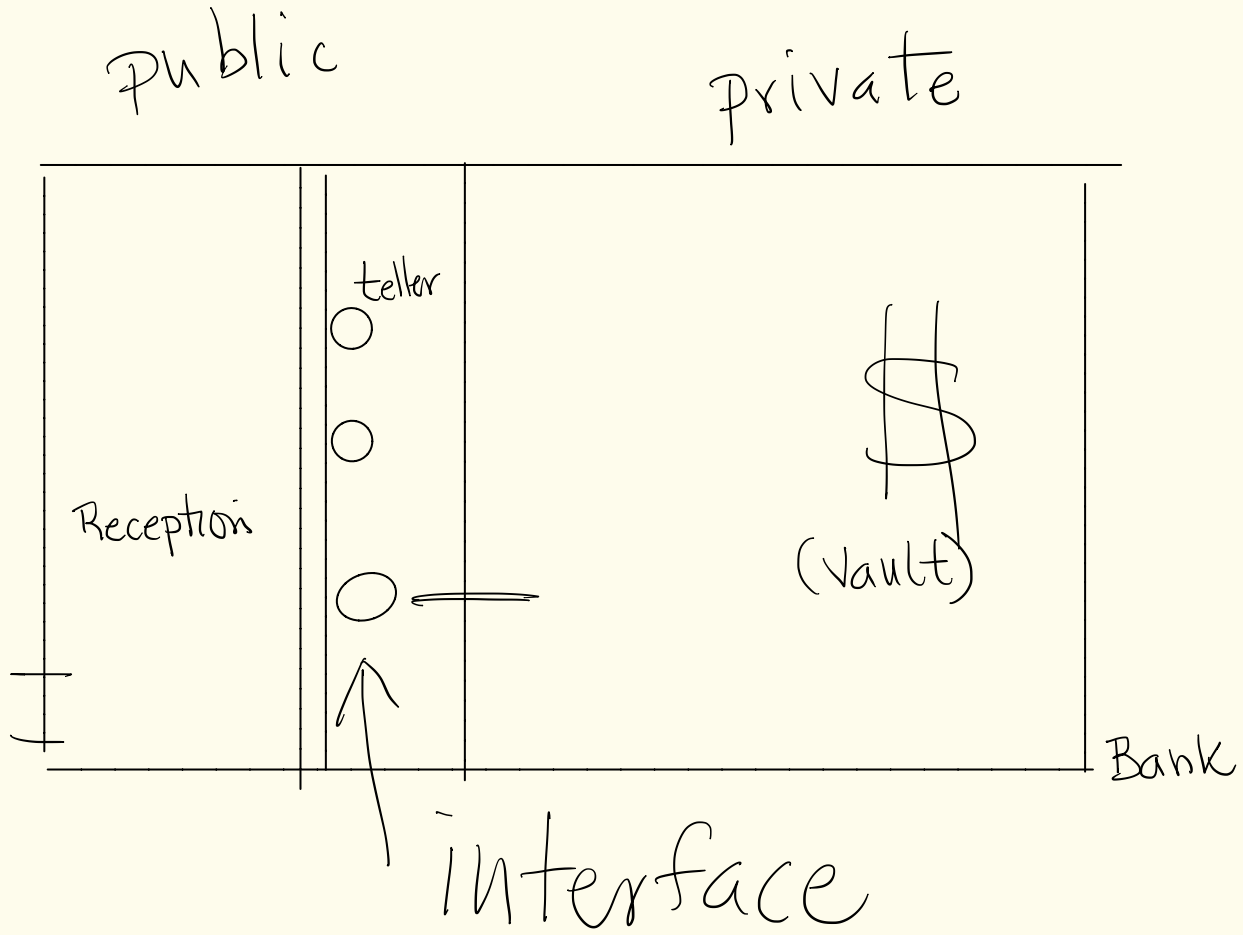
```
int b, c;
int a = 1;
if (foo() == 1) b = 2;
else b = "house";
c = a + b;
```

← compilation error

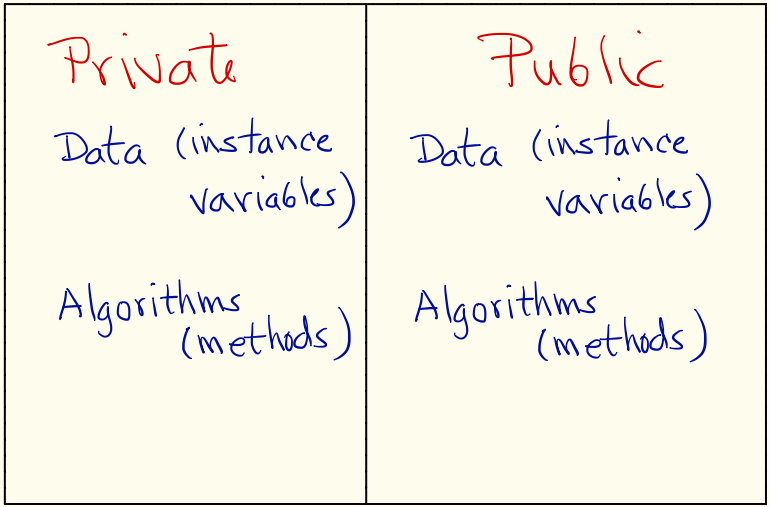
Software Life Cycle:

- Specification
- Design (Data, Algorithms)
- Implementation (Translation of design to a programming language)
- Testing and debugging


Interface
Public Part
Private part
Module
Information hiding, or Encapsulation


public
private
teller
Reception
(vault)
Bank
interface

Object


Private
Data (instance variables)
Algorithms (methods)
Public
Data (instance variables)
Algorithms (methods)

Objects have

- ***Data (properties of an object)***

- ***Algorithms*** (***or behaviours or actions***)

Objects have

- ***Data (properties of an object)***
  - In Java they are called ***attributes*** or ***fields*** or ***instance variables***
- ***Algorithms*** (***or behaviours or actions***)

Objects have

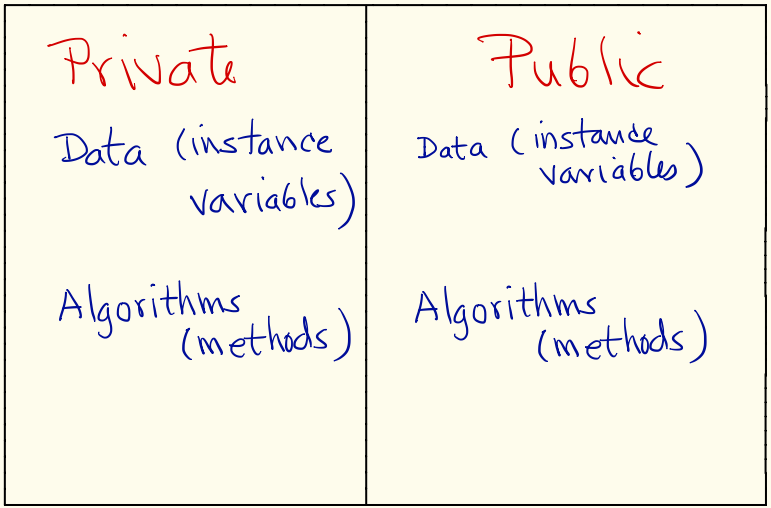
- ***Data (properties of an object)***
  - In Java they are called ***attributes*** or ***fields*** or ***instance variables***
- ***Algorithms*** (***or behaviours or actions***)
  - In Java they are implemented as ***methods*** (more specifically, ***instance methods***)

Objects have

- ***Data (properties of an object)***
  - In Java they are called ***attributes*** or ***fields*** or ***instance variables***
- ***Algorithms*** (***or behaviours or actions***)
  - In Java they are implemented as ***methods*** (more specifically, ***instance methods***)

Both can be private or public.

Object


Private
Data (instance variables)
Algorithms (methods)
Public
Data (instance variables)
Algorithms (methods)

Objects and Classes

Every object belongs to a specific ***class*.**

A **class** specifies the instance variables and methods of an object

Objects and Classes

Every object belongs to a specific ***class.***

A **class** specifies the instance variables and methods of an object, so a class definition consists of

- Instance variable declarations
  (also known as fields or attributes), and
- Method definitions

Example of a Java Class

```
public class Person {
    /* Attribute declarations */
    private String lastName;
    private String firstName;
    private String email;

    ...

    // Method declarations
    public String getEmail() {
                return email;
    }
    public void setEmail (String mail) {
                email = mail;
    }
}
```

Class must be stored in a file called Person.java

getter

setter

Objects and Classes

Every object belongs to a specific ***class.***

A **class** specifies the instance variables and methods of an object, so a class definition consists of

- Instance variable declarations
  (also known as fields or attributes), and
- Method definitions

A class definition must be stored in a file with the same name as the class and a .java extension.

Data Types

Java data types:

- primitive
- non-primitive or classes

Data Types

Java data types:

- primitive
- non-primitive or classes

There are 8 primitive Java data types:

boolean, char, byte, short, int, long, float, double

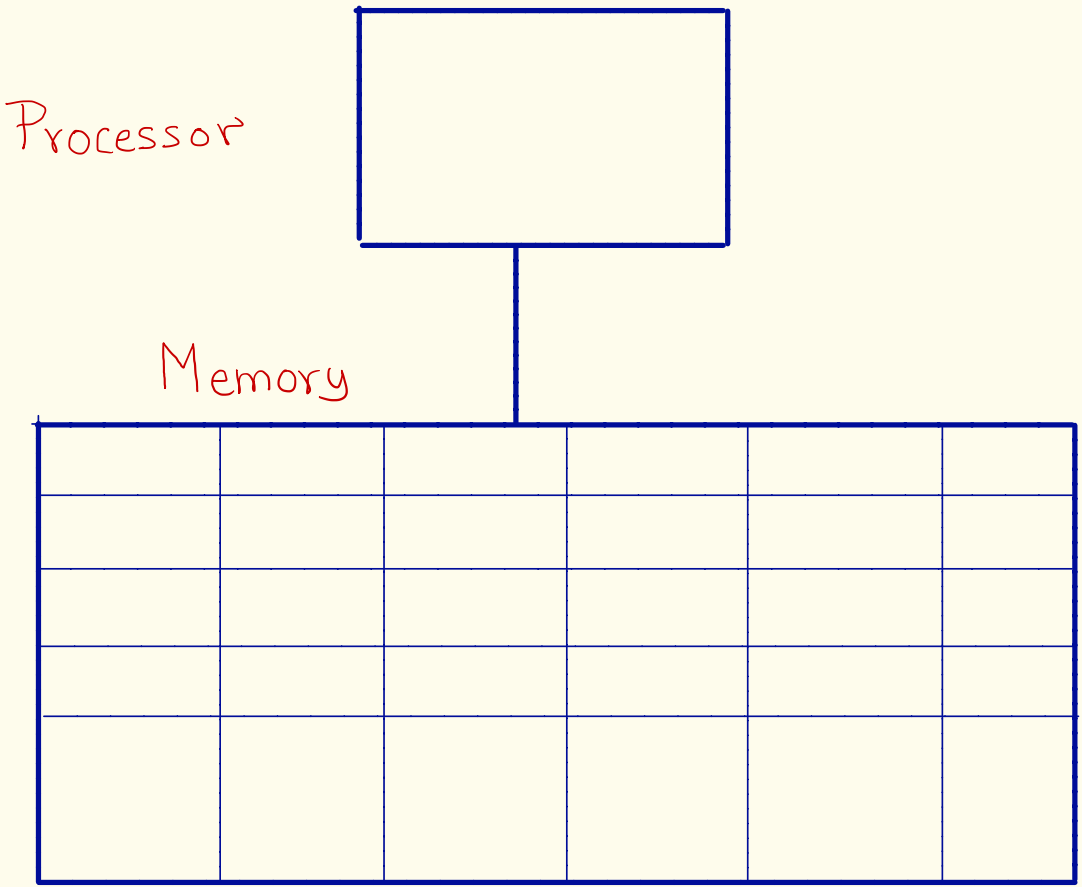
8bits 16 32 64 32 64

integer

real

2.7145

Computer Model



```
int a,b;
a=52;
b=a+1;
    52+1
Person p;
p = new Person();
```

32

Declaration

All variables must be initialized!

reference variables

Symbol table

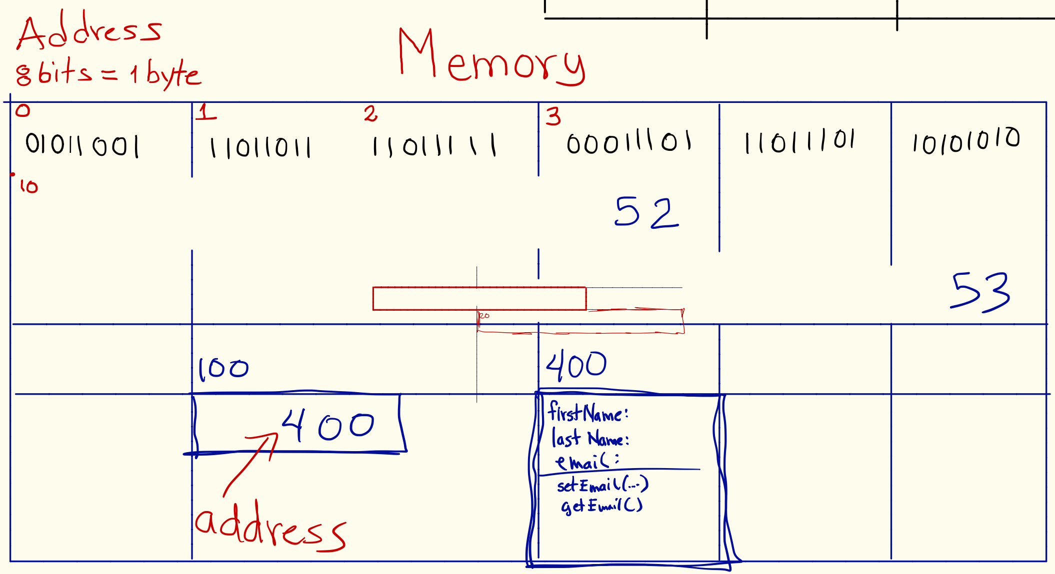
Var	Type	Address
a	int	**10**
b	int	20
p	Person	100

52 = 110100

2 | 52 = 26, remainder 0
2 | 26 = 13, remainder 0
2 | 13 = 6, remainder 1
2 | 6 = 3, remainder 0
2 | 3 = 1, remainder 1
2 | 1 = 0, remainder 1

↑ Converting 52 to binary

Memory



Data Types

- A variable of a primitive type stores a value.
- A variable of a non-primitive type stores and address.

or class

Example of a Java Class

```
public class Person {

    /* Attribute declarations */
    private String lastName;
    private String firstName;
    private String email;

    ...

    // Method declarations
    public String getEmail() {
                return email;
    }
    public void setEmail (String mail) {
                email = mail;
    }
}
```

java.lang

```
int a, b;
a = 52;
b = a+1;
Person p;
p = new Person();
p.setEmail("jd@uwo.ca");
p.setName("joe", "doe");
Person q;
q = new Person("joe", "doe", "jd@uwo.ca")
```

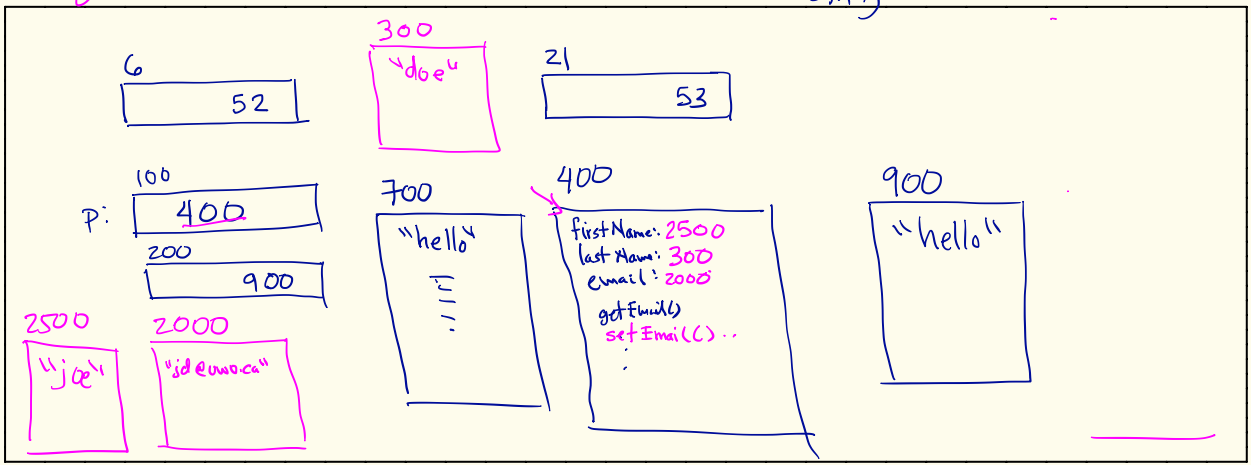
De-referencing

Symbol Table

var	type	Address
a	int	6
b	int	21
p	Person	100
s	String	200

=> Garbage collector

Memory



Constructors

A ***constructor*** is a special *method* that is called automatically when an object is created with the new operator.

Its purpose is to *initialize the attributes* of an object when the object is created

In Java a constructor has the same name as the class name

```
public class Person {

        /* Attribute declarations */
        private String lastName;              // last name
        private String firstName;             // first name
        private String email;                 // email address

          /* Constructor initializes the person's name and
             email address */
          public Person() {
                lastName = "";
                firstName = "";
                email = "";
          }

           public String getName() {
                  return firstName + " " + lastName;
           }

           public String getEmail() {
                  return email;
           }

           public void setEmail (String mail) {
                  email = mail;
           }

           public void setName(String newFirst, String newLast) {
                  firstName = newFirst;
                  lastName = newLast;
    }
```

```
/* Constructor initializes the person's name and
   email address */
public Person() {
        lastName = "";
        firstName = "";
        email = "";
}
public Person(String first, String last, String mail) {
        firstName = first;
        lastName = last;
        email = mail;
}
```

Is this allowed?

```
/* Constructor initializes the person's name and
   email address */
public Person() {
        lastName = "";
        firstName = "";
        email = "";
}
public Person(String first, String last, String mail) {
        firstName = first;
        lastName = last;
        email = mail;
}
```

Overloading: Two or more methods have the same name

```
/* Constructor initializes the person's name and
   email address */
public Person() {
        lastName = "";
        firstName = "";
        email = "";
}
public Person(String first, String last, String mail) {
        firstName = first;
        lastName = last;
        email = mail;
}
```

OK

Signature: Method name + number and types of parameters

```
/* Constructor initializes the person's name and
   email address */
public Person() {
        lastName = "";
        firstName = "";
        email = "";
}
public Person() {
        firstName = "John";
        lastName = "Doe";
        email = "jd@mail";
}
```

Invalid

Signature: Method name + number and types of parameters

```
public class Person {

    /* Attribute declarations */
    private String lastName;          // last name
    private String firstName;         // first name
    private String email;             // email address

    /* Constructor initializes the person's name and
       email address */
    public Person() {
        lastName = "";
        firstName = "";
        email = "";
    }
    public Person(String first, String last, String mail) {
        firstName = first;
        lastName = last;
        email = mail;
    }
```

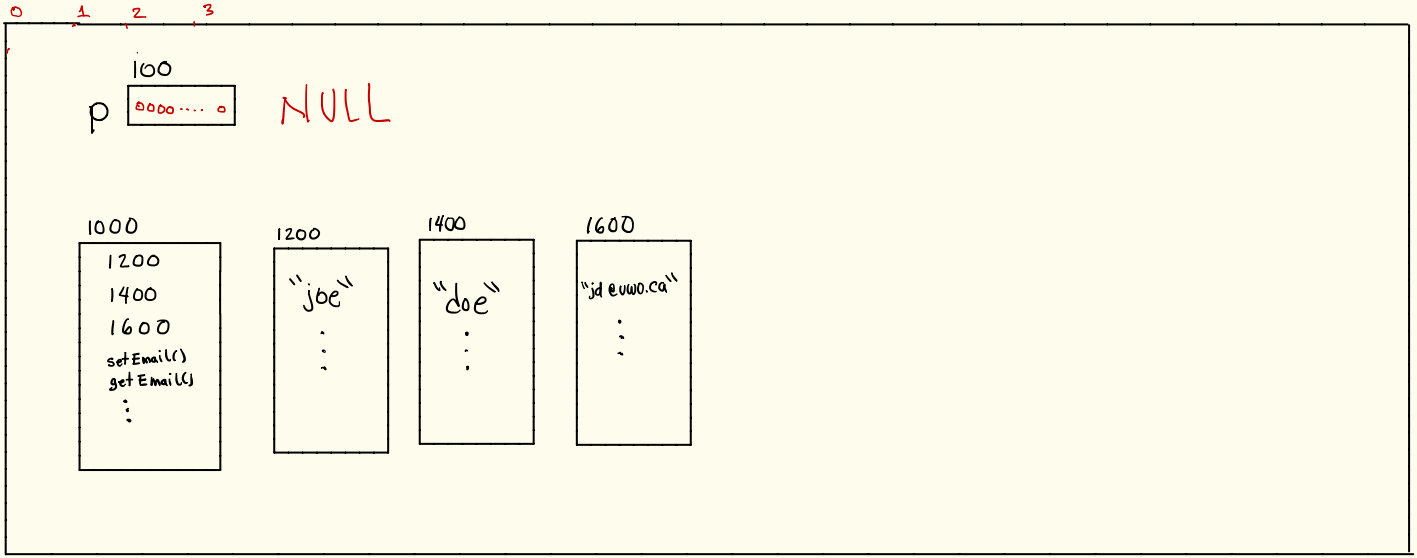
formal parameters

```
    public String getName() {
        return firstName + " " + lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail (String mail) {
        email = mail;
    }

    public void setName(String newFirst, String newLast) {
        firstName = newFirst;
        lastName = newLast;
```

Symbol Table

Var	Type	Value
p	Person	100

```
Person p;
p.getEmail();  ← Would throw a Null pointer exception
p = new Person("joe", "doe", "jd@uwo.ca");
p.getEmail();
```

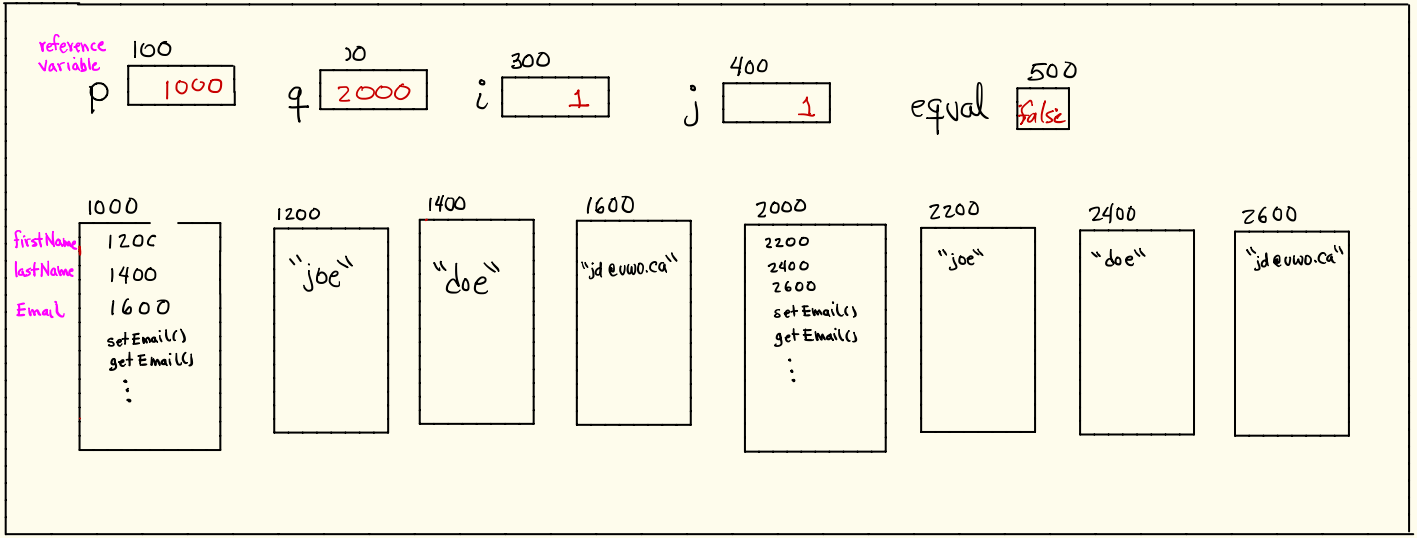


```
Person p, q;
int i,j;
boolean equal;

i=1;
j=1;
if (i==j) equal=true;
else equal=false;
p= new Person("joe", "doe", "jd@uwo.ca");
q= new Person("joe", "doe", "jd@uwo.ca");
if (p==q) equal=true;
else equal=false;
```

actual parameters

Symbol Table

Var	Type	Value
p	Person	100
q	Person	200
i	int	300
j	int	400
equal	boolean	500

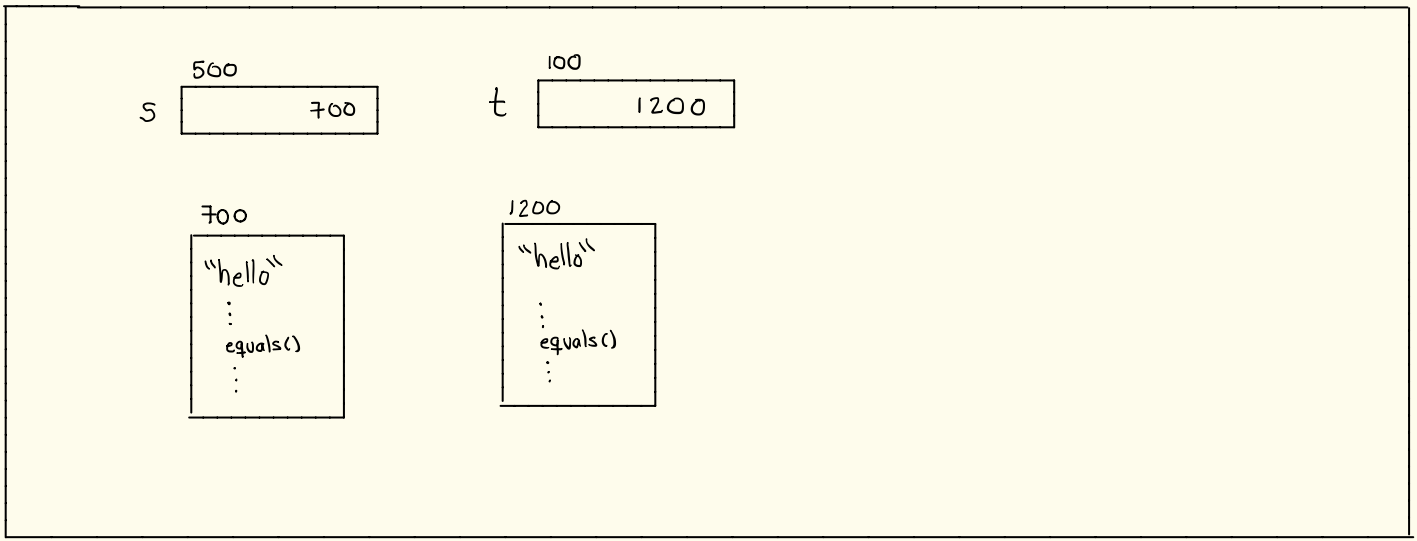


```
String s, t;
s = new String("hello");
t = "hello";
if(s.equals(t))
    System.out.println("equal");
else System.out.println("different");
```

Symbol Table

Var	Type	Value
s	String	50
t	String	100

Memory

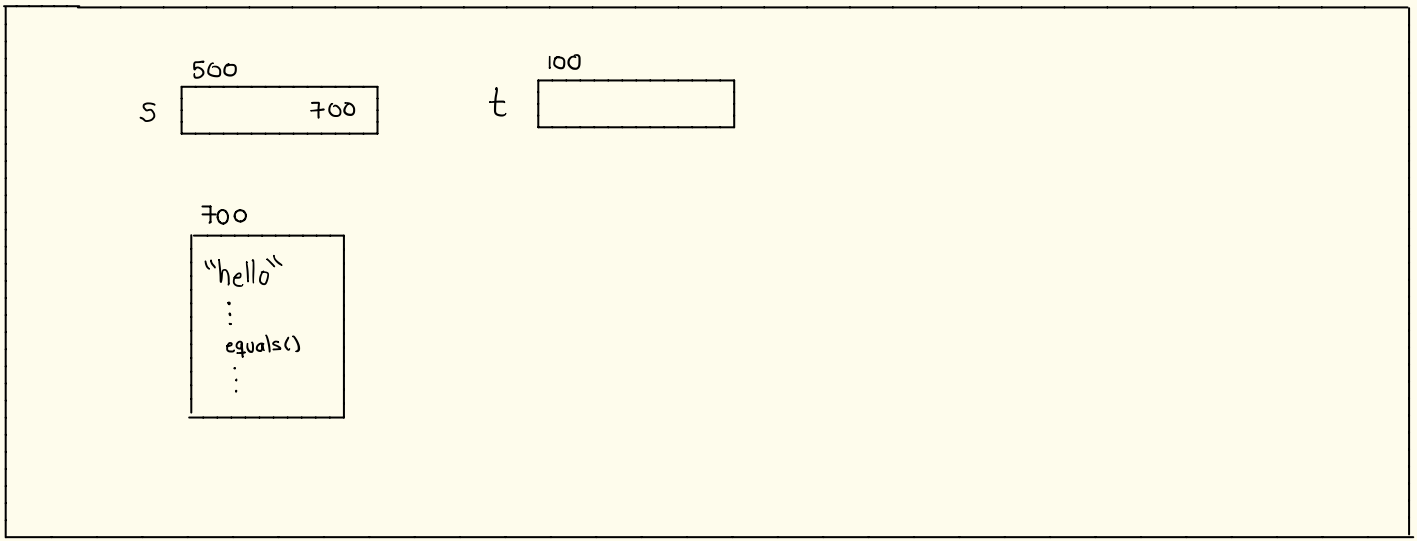


```
String s, t;
s = "hello";
t = "hello";
if (s==t)
    System.out.println("equal");
else System.out.println ("different");
```

Symbol Table

Var	Type	Value
s	String	50
t	String	100

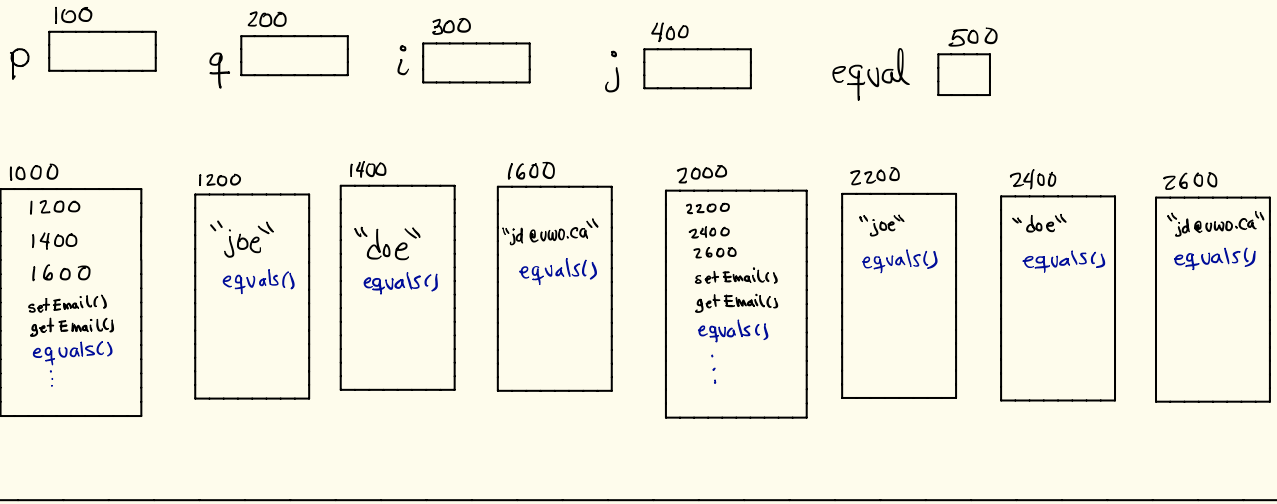
Memory



```
Person p, q;
int i,j;
boolean equal;

i=1;
j=1;
if (i==j) equal=true;
else equal=false;
p= new Person("joe", "doe", "jd@uwo.ca");
q= new Person("joe", "doe", "jd@uwo.ca");
if (p==q)    equal=true;
else equal=false;
```

Symbol Table

Var	Type	Value
p	Person	100
q	Person	200
i	int	300
j	int	400
equal	boolean	500



```
public class Person {

        /* Attribute declarations */
        private String lastName;              // last name
        private String firstName;             // first name
        private String email;                 // email address

          /* Constructor initializes the person's name and
             email address */
          public Person() {
                lastName = "";
                firstName = "";
                email = "";
          }

           public String getName() {
                  return firstName + " " + lastName;
           }

           public String getEmail() {
                  return email;
           }

           public void setEmail (String mail) {
                  email = mail;
           }

           public void setName(String newFirst, String newLast) {
                  firstName = newFirst;
                  lastName = newLast;
    }
```

```
/**
 * equals method determines whether two Person objects
  * have the same name
 * @param other:Person object that this is compared to
 * @return true of they have the same first name and last
 *  name, false otherwise
 */
public boolean equals(Person other){
        if (this.firstName.equals(other.firstName) &&
                        this.lastName.equals(other.lastName))
                return true;
        else
                return false;
}
```

```
/**
 * equals method determines whether two Person objects
  * have the same name
 * @param other:Person object that this is compared to
 * @return true of they have the same first name and last
 *  name, false otherwise
 */
public boolean equals(Person other){
        if (this.firstName.equals(other.firstName) &&
                        this.lastName.equals(other.lastName))
                return true;
        else
                return false;
}
```

```
/**
 * equals method determines whether two Person objects
  * have the same name
 * @param other:Person object that this is compared to
 * @return true of they have the same first name and last
 *  name, false otherwise
 */
public boolean equals(Person other){
        if (this.firstName.equals(other.firstName) &&
                        this.lastName.equals(other.lastName))
                return true;
        else
                return false;
}
```

```
/**
 * equals method determines whether two Person objects
  * have the same name
 * @param other:Person object that this is compared to
 * @return true of they have the same first name and last
 *  name, false otherwise
 */
public boolean equals(Person other){
        if (this.firstName.equals(other.firstName) &&
                        this.lastName.equals(other.lastName))
                return true;
        else
                return false;
}
```

```
    /**
     * toString method returns a string representation of the person
     * @return string with first name and last name, email address
     */
    public String toString() {
            String s = firstName + " " + lastName + "\t" + email;
            return s;
    }
}
```