JAVA Programming

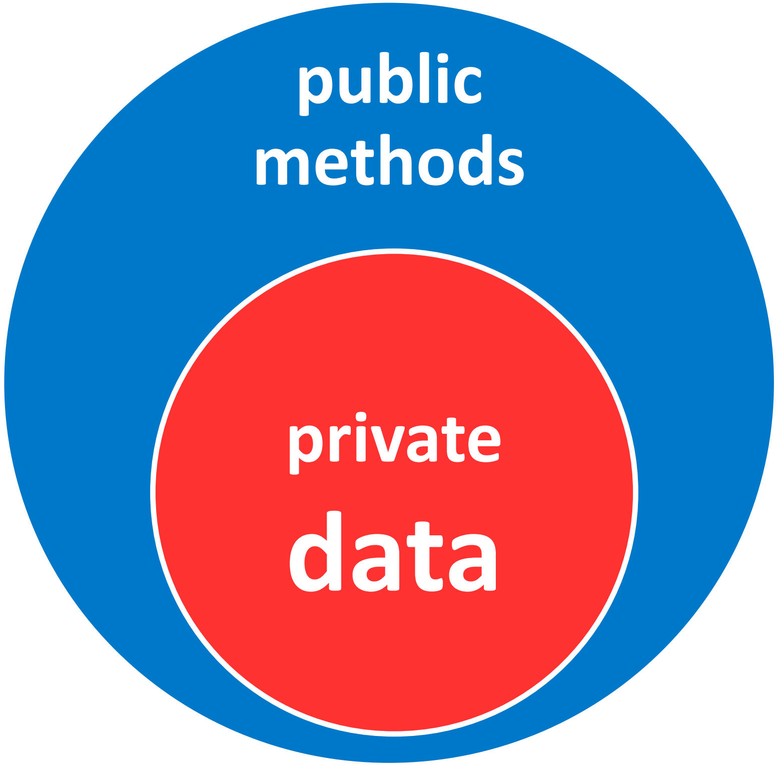
Object Oriëntation

Overview

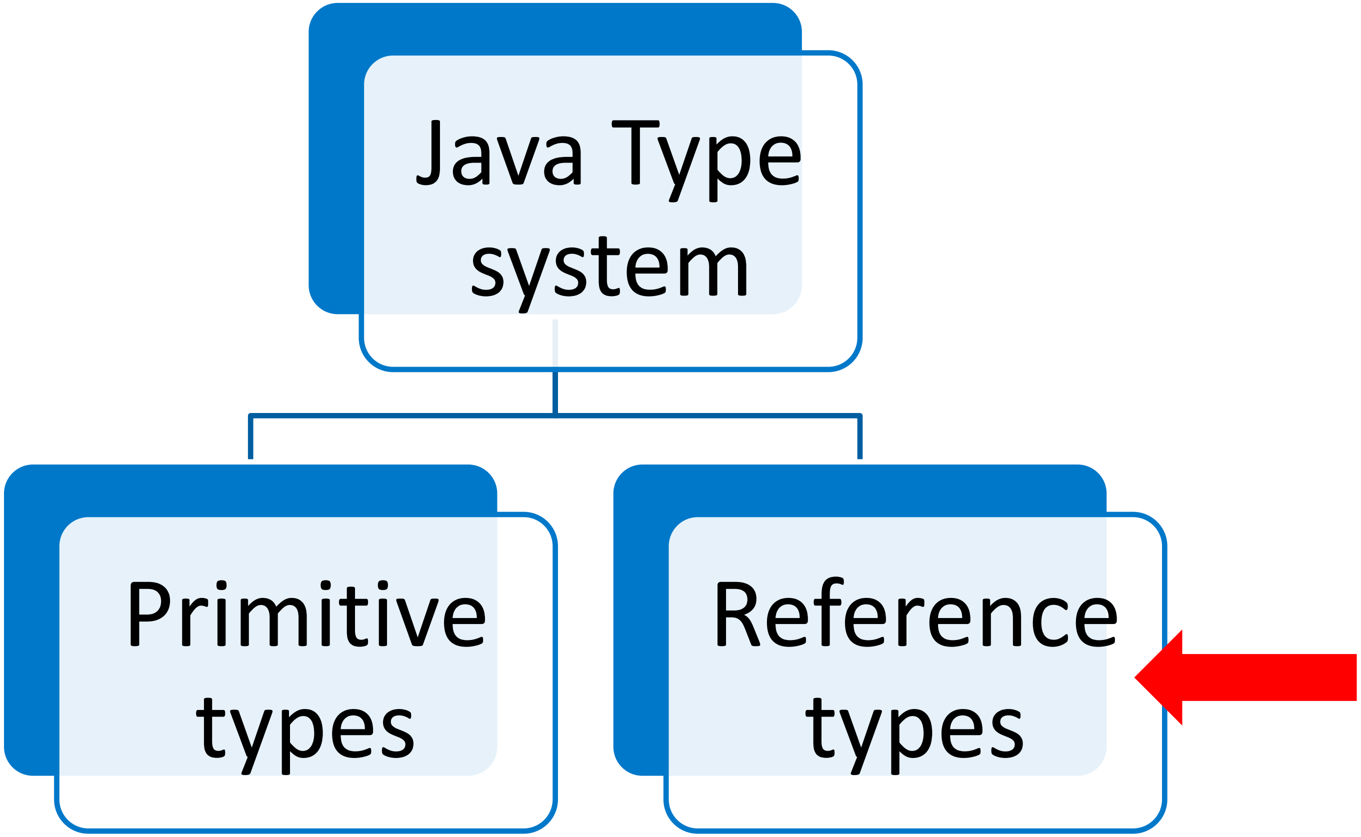
- Reference Types
- The this keyword
- Java memory model
- Static Data
- Default values and null
- Final fields and variables

Reference types

- encapsulation
- information hiding
- naming conventions (Java Beans standard)



Reference types



Reference types

- Java needs a “blueprint”
- We must specify this object in Java terms.
- How can we do this?

Reference types

- How can we specify the blue print for a Client object?
  - A client has data
  - A client has functionality

Client	
int	clientId
String	name

Class: Blue print keyword

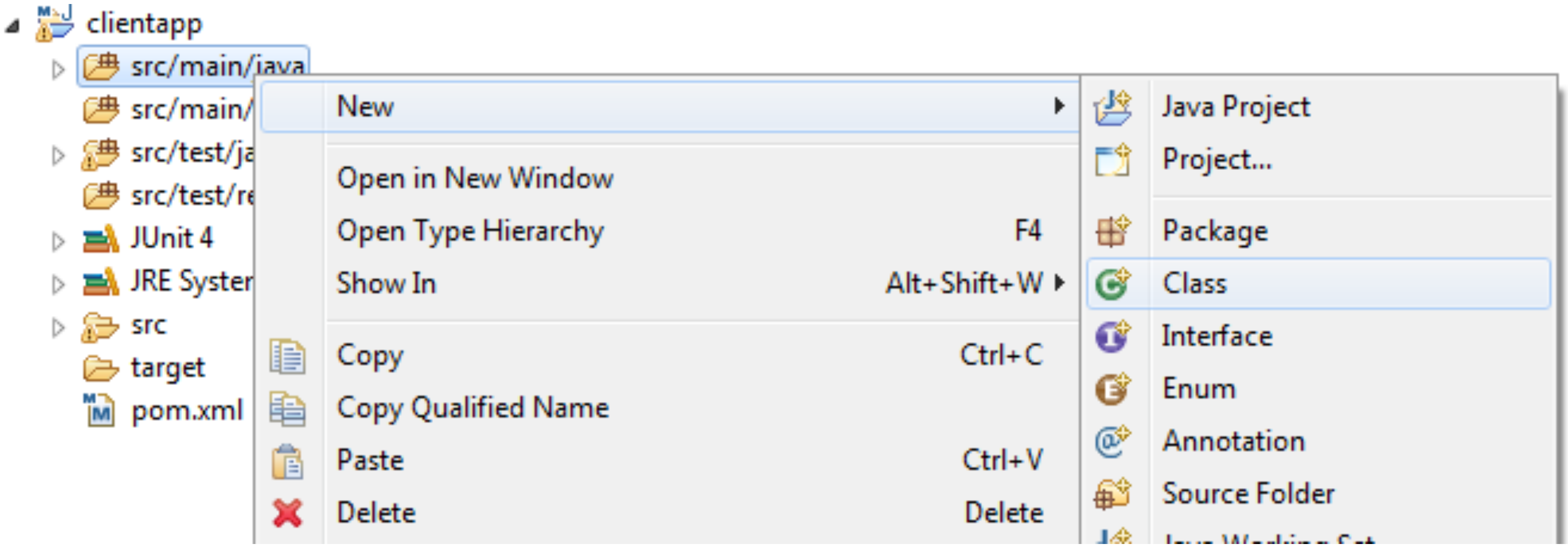
Uppercase

```
public class Client {

   public int clientId;

   public String name;
}
```

lowercase

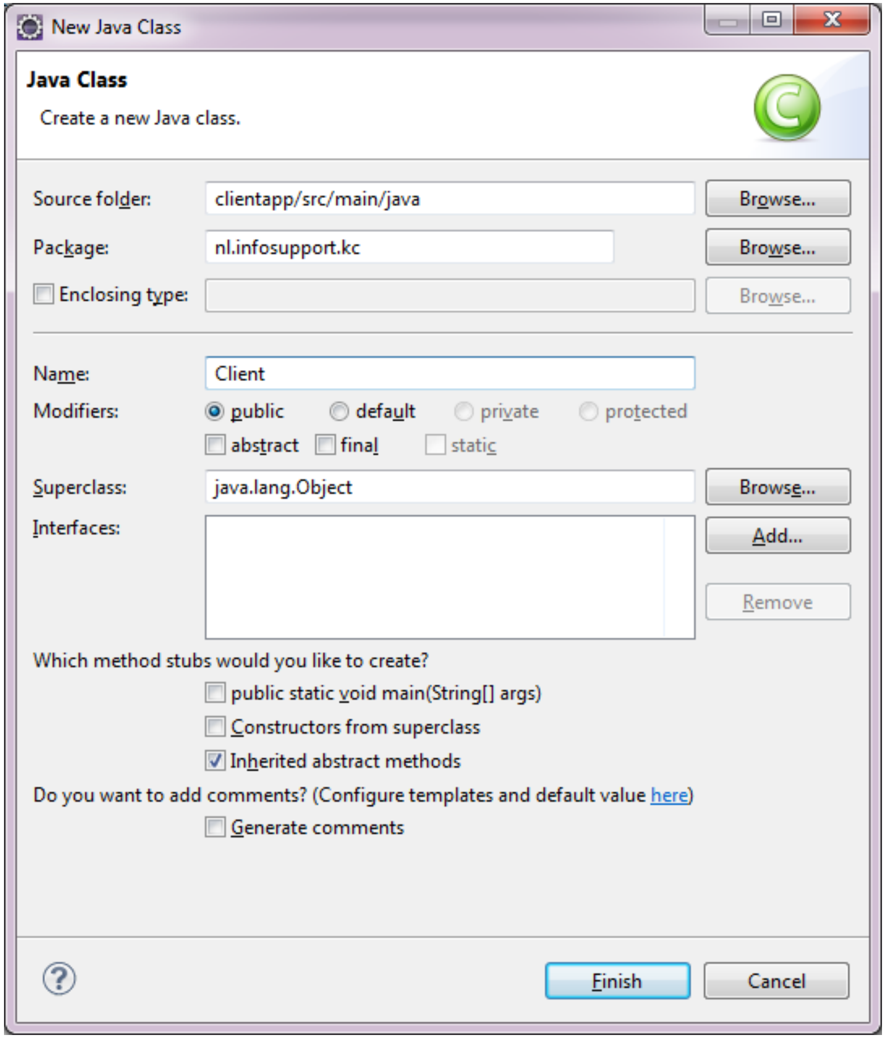
Reference types

- In Eclipse:



Reference types

- In Eclipse:



Java Class

Create a new Java class.

Source folder: clientapp/src/main/java

Package: nl.infosupport.kc

□ Enclosing type:

Name: Client

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Reference types

- How can I use my client “blueprint”?
- Declaring a class (blue print) variable does not create an object
  - Use the **new** operator to create an object

Reference types

- Java needs a “blueprint” – ***class*** keyword
- We can create a Client using it’s ***constructor***
- Creation of objects → ***instantiate***

Reference types

- Create a **reference** to the client.

```
Client myClient = new Client();
myClient.name = “name of client”;
```

- Based on the same “blue print” we can create an unlimited number of Client objects.

```
Client client1 = new Client();
client1.name = “Jan”;
Client client2 = new Client();
client2.name = “Leopold”;
```

Reference types

- Creation statement:

Variable name

Constructor call

```
Client myClient = new Client();
```

Type

- Used in an Array

```
Client[] myClients = new Client[10];
myClients[0] = new Client();
```

Reference types

- Take a deeper look into our blue print

```
public class Client {

  public int clientId;

  public String name;

}
```

No data hiding, class members are accessible to the outside world.

Reference types

- Take a deeper look into our blue print

```
public class Client {

  ~~public~~ private int clientId;

  ~~public~~ private String name;

}
```

Reference types

- Take a deeper look into our blue print

```
public class Client {

  ~~public~~ private int clientId;

  ~~public~~ private String name;

}
```

- Fields not visible

```
Client client1 = new Client();
client1.name = “Klaas”;
```

Compilation error: not visible

- To access the data, create getters and setters.

Reference types

- Getters for read access
  - String getName()
  - int getId()
- Setters for write access
  - void setName(String newName);
  - void setId(int newId);

Reference types

- Getters and setters

```
public class Client {
 private String name;
 private int id;

 public String getName() {
   return name;
 }

 public void setName(String newName) {
   name = newName;
 }
```

Private fields describe Inaccessible state

Public methods Describe accessible behaviour

starts with ***get***

Property name, starts with uppercase

Property name, starts with Uppercase

starts with ***set***

Reference types

- Data hiding

```
public class Client {
 private String name;
 private int id;

 public String getName() {
   return name;
 }

 public void setName(String newName) {
   name = newName;
 }
```

newName is now: “Jan”

```
Client client1 = new Client();
client1.setName(“Jan”);
```

The “this” keyword

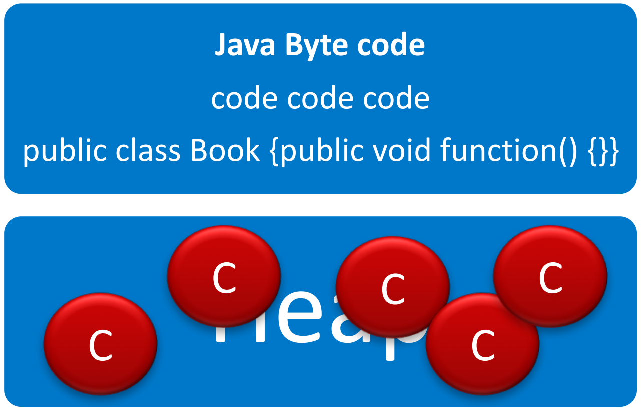
- can be used in non-static methods
  - within a class it represents a reference to the current instance
  - Useful when identifiers from different scopes clash

```
public class Client {

  private String name;

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

Java memory model

- Code is loaded once
- Data for each object an instance on the heap

```
public String getName() {
  return name;
}
```



Java Byte code

code code code

public class Book {public void function() {}}

Heap

Five instances in memory
but the code only once

Static Data

- Look at the following code example

```
public class Client {

  private String type;

  private int years;

  public void setYears(int years) {
    if (years < 33) {
      // throw exception
    } else {
      this.years = years;
    }
  }
...
```

What’s this
magic number?

Static Data

- Create a descriptive variable for 33

```
public class Client {
  private int MINIMUM_YEARS = 33;

  private String type;

  private int years;

  public void setYears(int years) {
    if (years < MINIMUM_YEARS) {
      // throw exception
    } else {
      this.years = years;
    }
  }
...
```

Static Data

- With 100.000 Client objects you now have 100.00 int’s MINIMUM_YEARS in memory

- Therefore static members exist.

Static Data

- Create a descriptive variable for 33

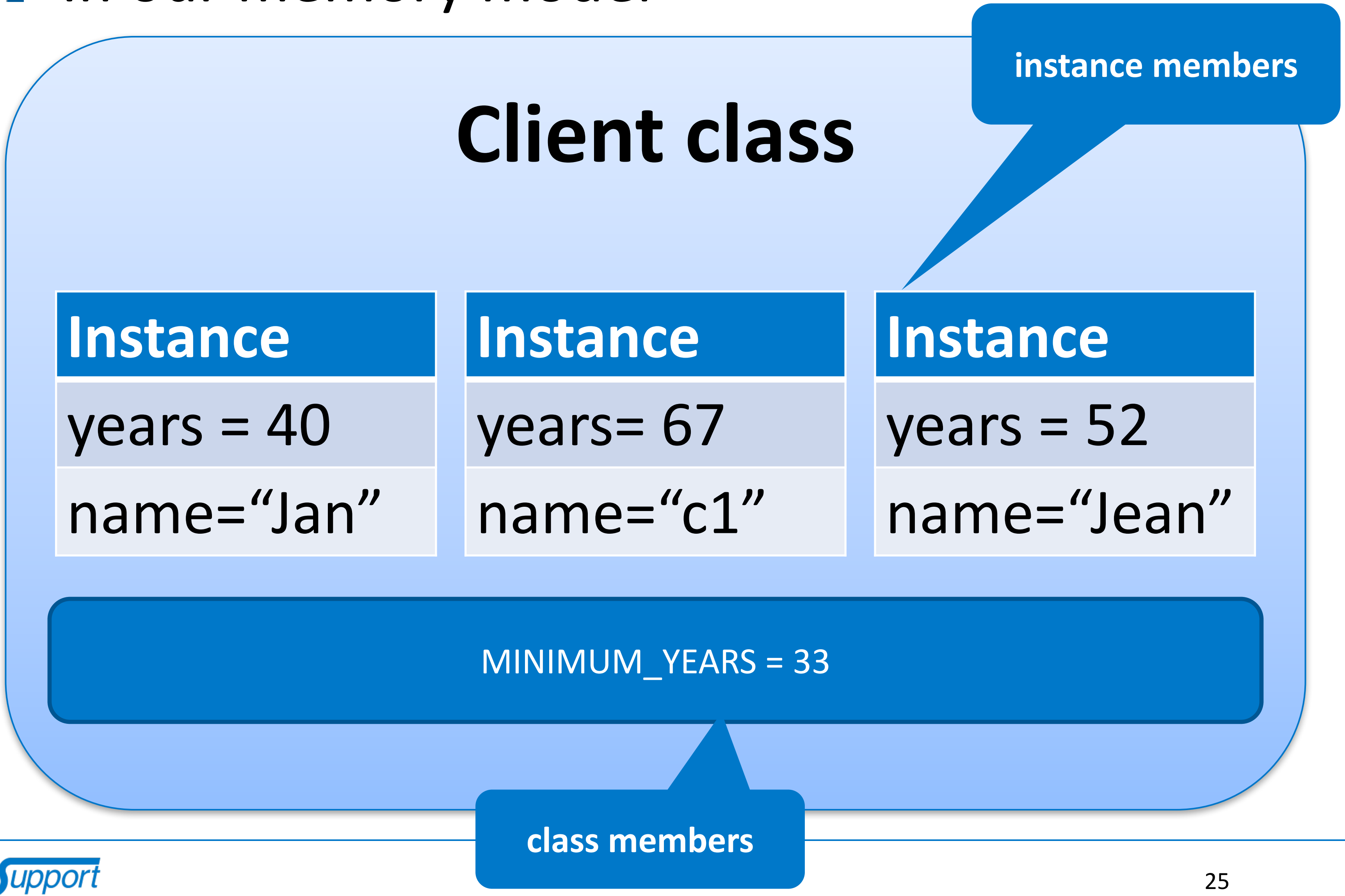
```
public class Client {
  private static int MINIMUM_YEARS = 33;

  private String type;

  private int years;

  public void setYears(int years) {
    if (years < MINIMUM_YEARS) {
      // throw exception
    } else {
      this.years = years;
    }
  }
...
```

UPPERCASED

Changed to static

Static Data

- In our memory model



instance members

Client class

Instance	Instance	Instance
years = 40	years= 67	years = 52
name=“Jan”	name=“c1”	name=“Jean”

MINIMUM_YEARS = 33

class members

Static

- We can also create static methods

```
public class Client {

  public static int MINIMUM_YEARS = 33;

  public static int getMINIMUM_YEARS() {
    return MINIMUM_YEARS;
  }

  public static void setMINIMUM_YEARS(int mINIMUM_YEARS) {
    MINIMUM_YEARS = mINIMUM_YEARS;
  }
```

Keyword this can not be used here

Static

- How to refer to the MINIMUM_YEARS?

```
@Test
public void test() {

 Client.MINIMUM_YEARS;
 Client.setMINIMUM_YEARS(43);
}
```

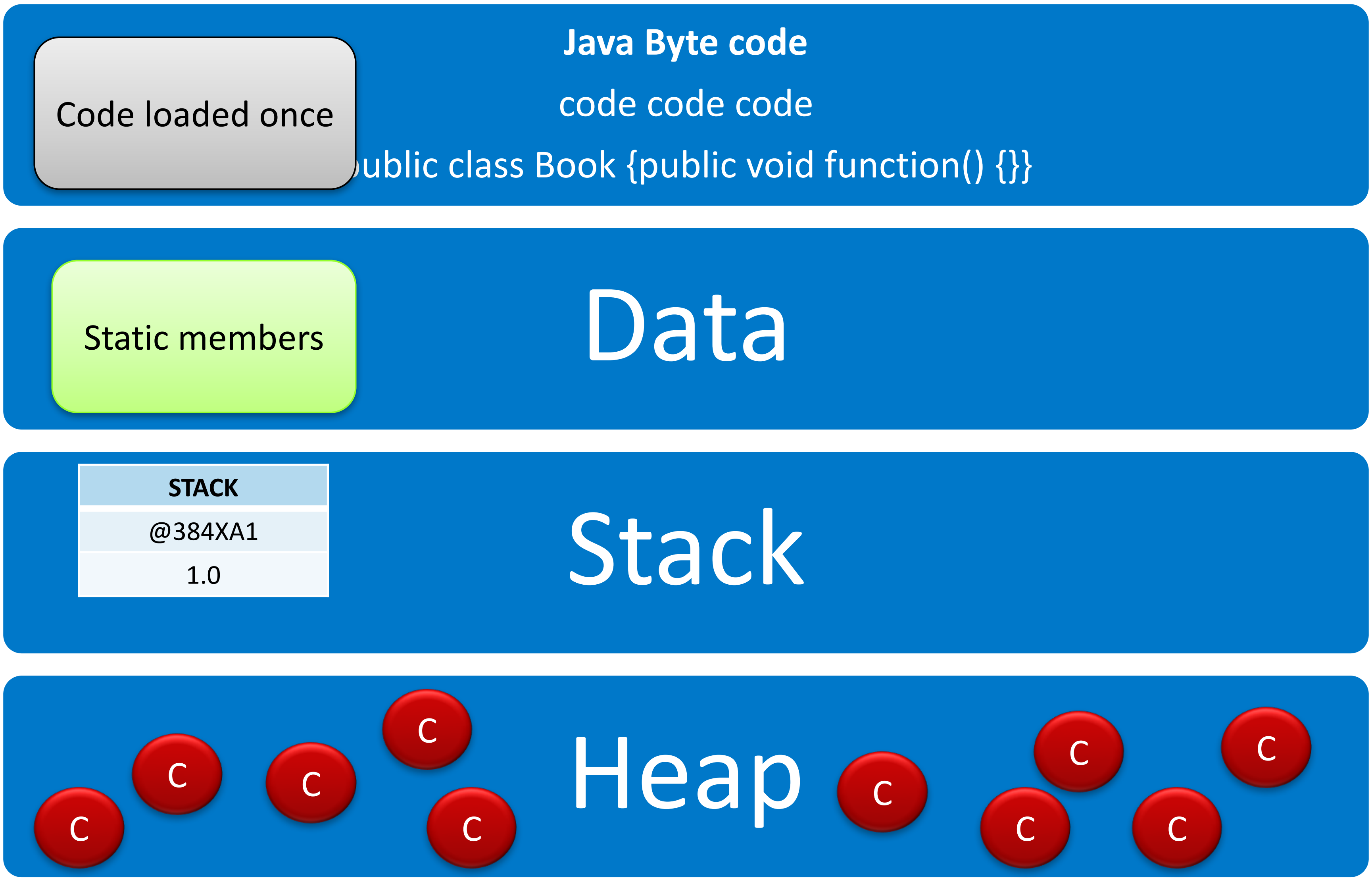
- Refer by Class, not instance

Static

- Examples of static members

```
int sum = Math.sum(1, 4);
double abs = Math.abs(-100.0);
Double pi = Math.PI;
```

Static Data



Code loaded once

Java Byte code

code code code

ublic class Book {public void function() {}}

Static members

Data

STACK
@384XA1
1.0

Stack

Heap

Default values and null

- For ‘reference to nothing’ Java has a special keyword: **`null`**

Data type	Default value
byte, short, int, long	0
boolean	false
char	‘<space> ‘
double	0.0
float	0.0
Object	null

Default values and null

- Only class members and attributes have a default value

```
public class Client {

  public void getClient() {
      long clientId;
      System.out.println(“Default clientId: ” + clientId);
      ...
  }
}
```

Does not compile: **The local variable clientId may not have been initialized**

Default values and null

- Only class members and attributes have a default value

```
public class Client {

  long clientId;

  public void getClient() {
      System.out.println(“Default clientId: ” + clientId);
      ...
  }
}
```

OK: The field clientId is initialized to its default value.

Final Fields and Variables

- Final fields and variables
  - Can only be assigned a value once
    - Declare and assign a value
    - If not assigned a value at declaration, a value can only be assigned in the constructor
  - Not same as a constant
    - Constant: value is known compile time
    - Final field: value is only known at run time

Lab : Object Oriëntation