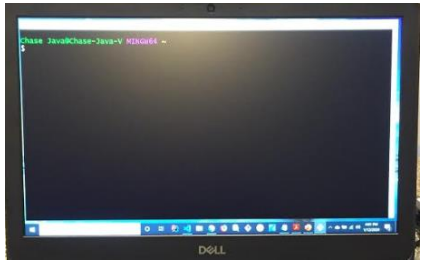


Cohort Laptop

Java victoria-hinkle-moore

pull upstream master

push origin master

- Submit your exercise work

Bitbucket

JPMC Tech Start

Repositories

Filter by: All Public

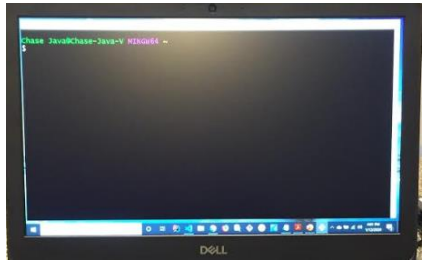
Repository

Java java-main

Java victoria-hinkle-moore

fork

Cohort Repo: first-lastname



Instructor Laptop

Java java-main

push origin master

Daily Git Commands

- Execute the git commands from the directory top of your repository
  - Current directory should be your-name folder inside your workspace folder

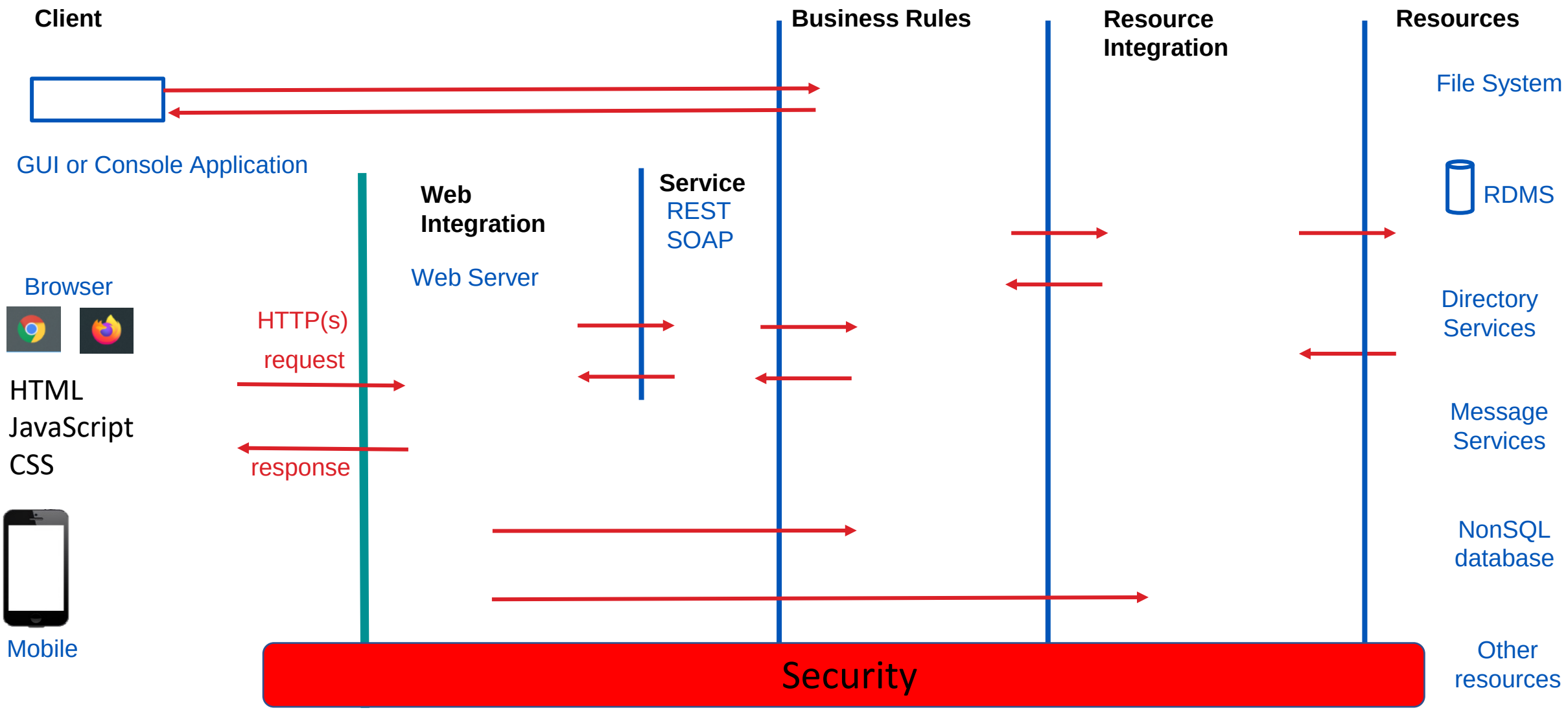
- Git Commands to **Pull** Daily Work From BitBucket
  - git pull upstream master

- Git Commands to **Push** Your Work To BitBucket
  - git add -A
  - git commit -m "with message"
  - git push origin master

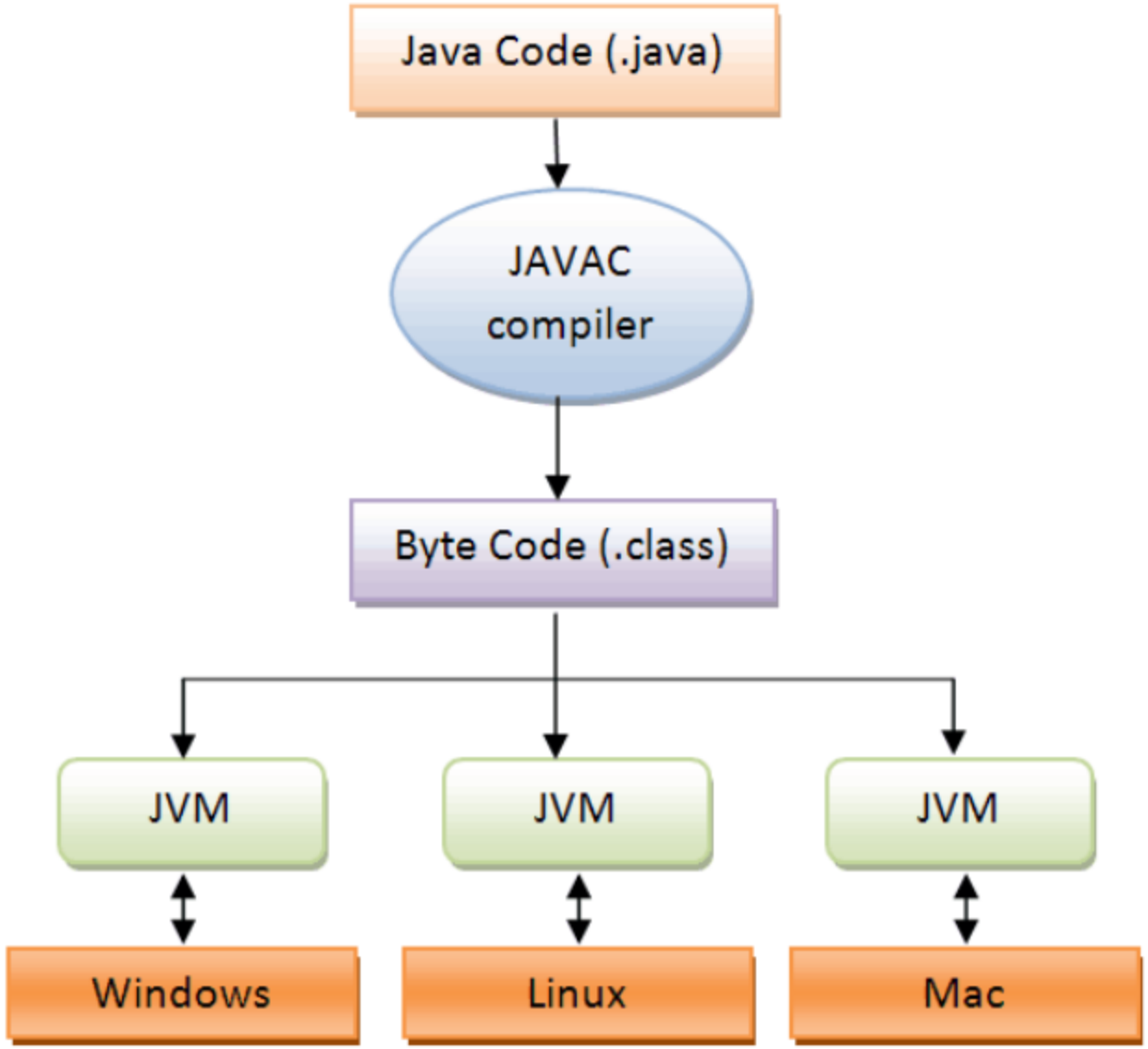
Some Basic Things to Remember

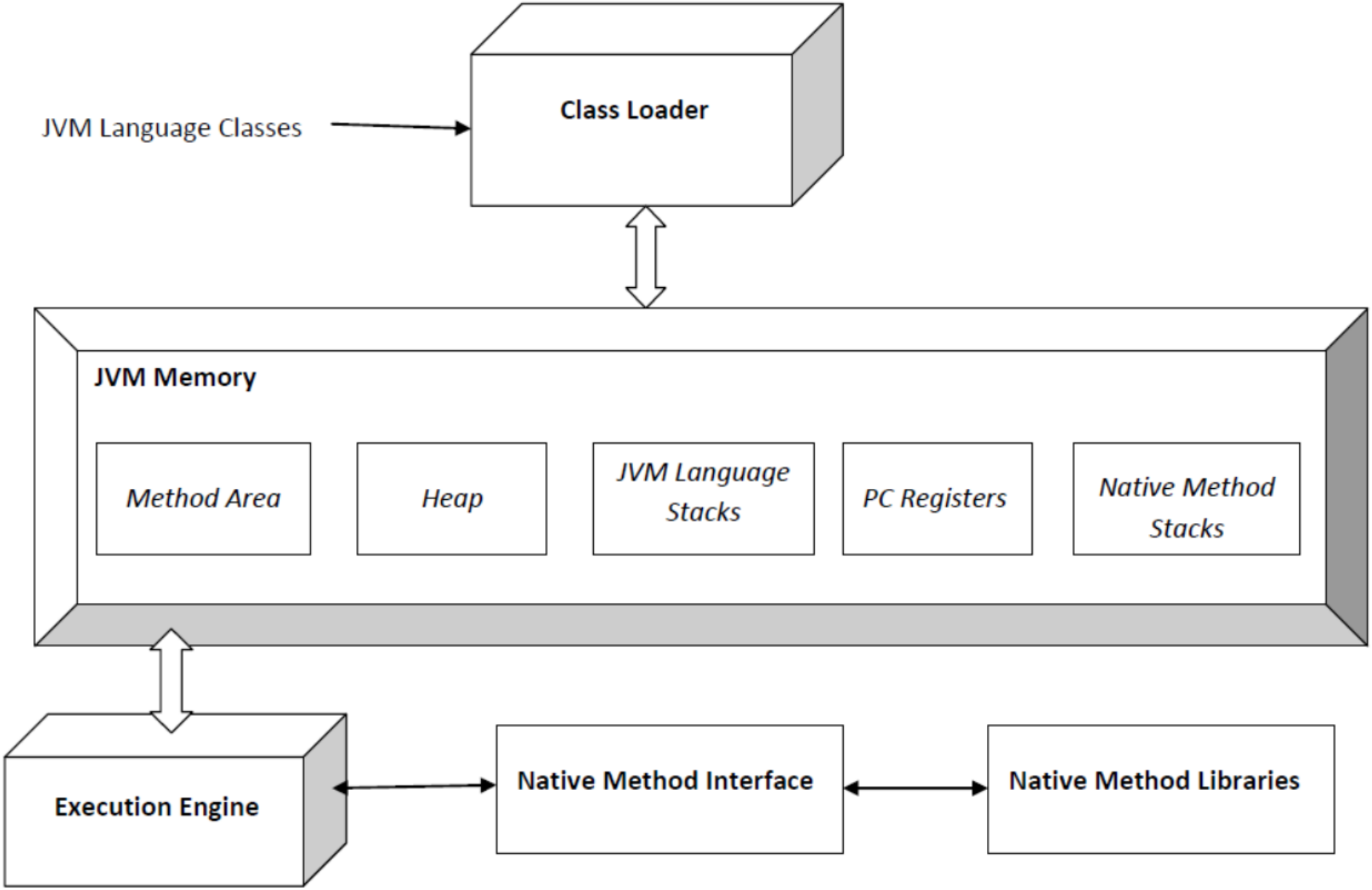
1. Only 2 general datatypes: Value (Primitives – stores the actual value) and Reference (all others – store a reference to the allocated data type in the Heap)
2. Any variable defined inside a method (this includes method parameters) is a local variable and will be allocated on the Stack in memory – if the datatype of the variable is a primitive the value will be stored, if the datatype is a reference a reference (address in the heap) to the value will be stored
3. A class may contain fields, methods, and constructors (same name as class with parameters to initialize object)
   - As a general rule with entity type classes (Customer) class is public, fields are private, constructors and methods are public
4. Method Overloading – same class, same name, unique parameter list (compiler figures out which method to call)
5. Method Overriding – super class, same name, same parameters (new implementation in subclass)
6. this. – field or method in same class this( ) – constructor call in same class
7. super. – field or method in super class super( ) – constructor call in super class
8. Concrete class – all methods implemented (must be concrete to create new object)
9. Abstract class – at least 1 abstract method (may contain concrete methods for inheritance) (either declare class abstract or implement all abstract methods)
10. Interface – concept all methods implicitly abstract (used to share common methods between classes not already related through inheritance). Good for code extensibility and loose coupling.
11. static modifier on field (single allocation for all instances) and methods accessed through ClassName.

Application Architecture



Authentication, Authorization, Encryption, CORS, Injection, …


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac


JVM Language Classes
Class Loader
JVM Memory
Method Area
Heap
JVM Language Stacks
PC Registers
Native Method Stacks
Execution Engine
Native Method Interface
Native Method Libraries

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Logical operator table

The following table sums up the different logical operators:

`A`	`B`	`!A`	`A && B`	`A || B`	`A ^ B`
TRUE	**TRUE**	FALSE	TRUE	TRUE	FALSE
TRUE	**FALSE**	FALSE	FALSE	TRUE	TRUE
FALSE	**TRUE**	TRUE	FALSE	TRUE	TRUE
FALSE	**FALSE**	TRUE	FALSE	FALSE	FALSE

Anatomy of a Method

return datatype
- primitive
- reference (Classname, …)
- void

method name

```
public int returnNotOne(int number, Customer customer)
{
        // statements – end with a semi colon
        // blocks – conditional and looping
        return number + 1;
}
```

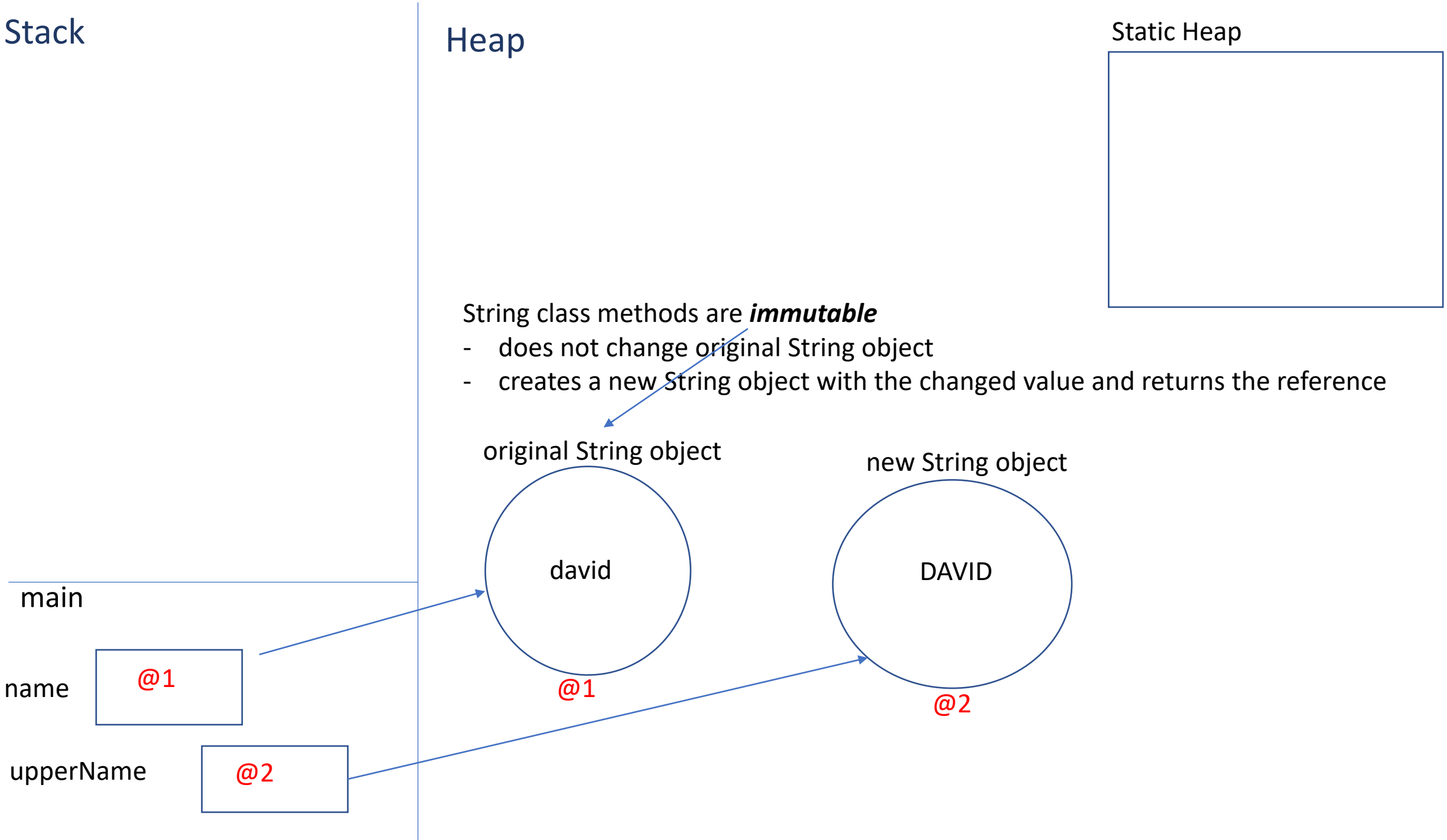
method arguments

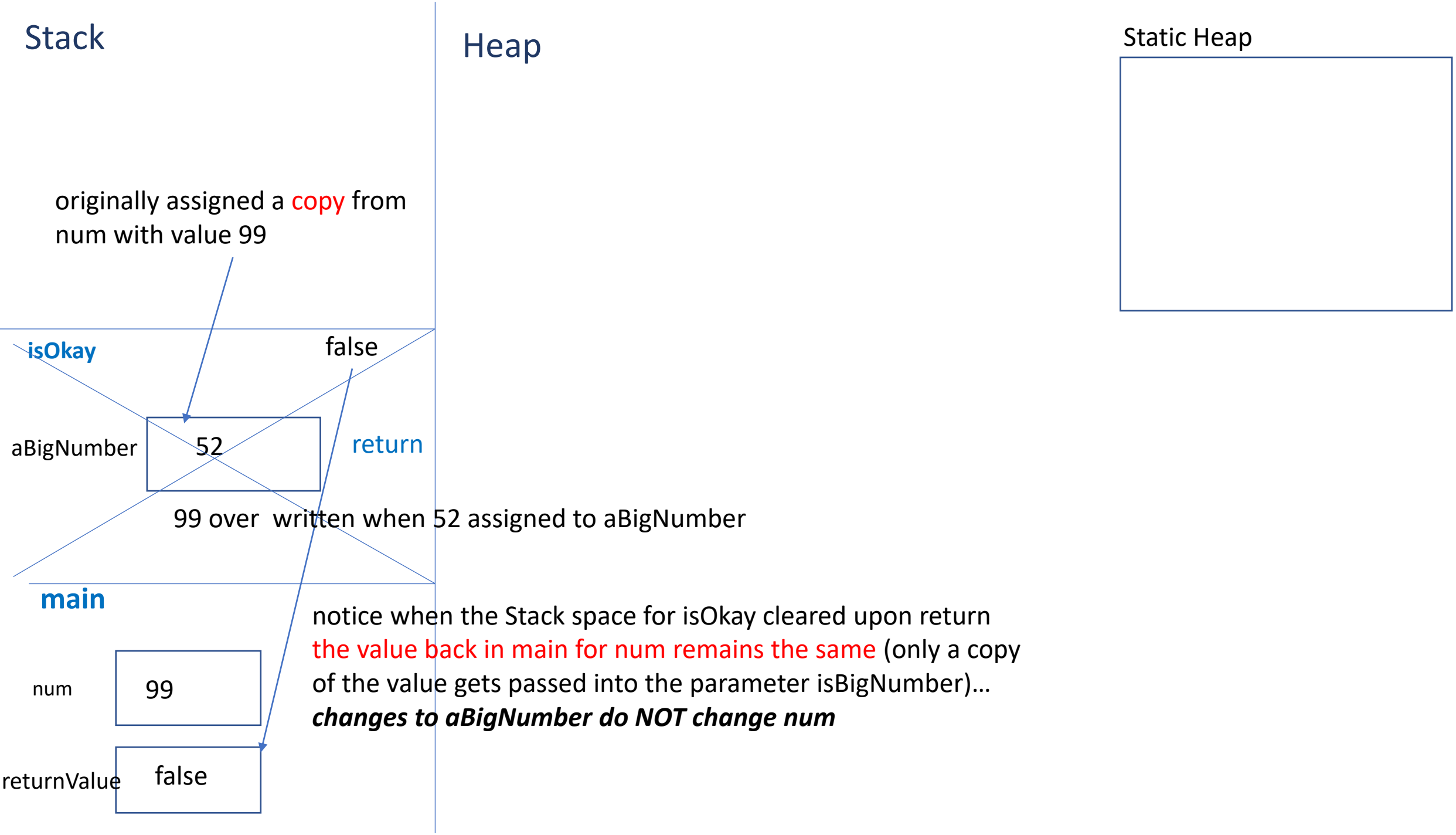
datatype argName

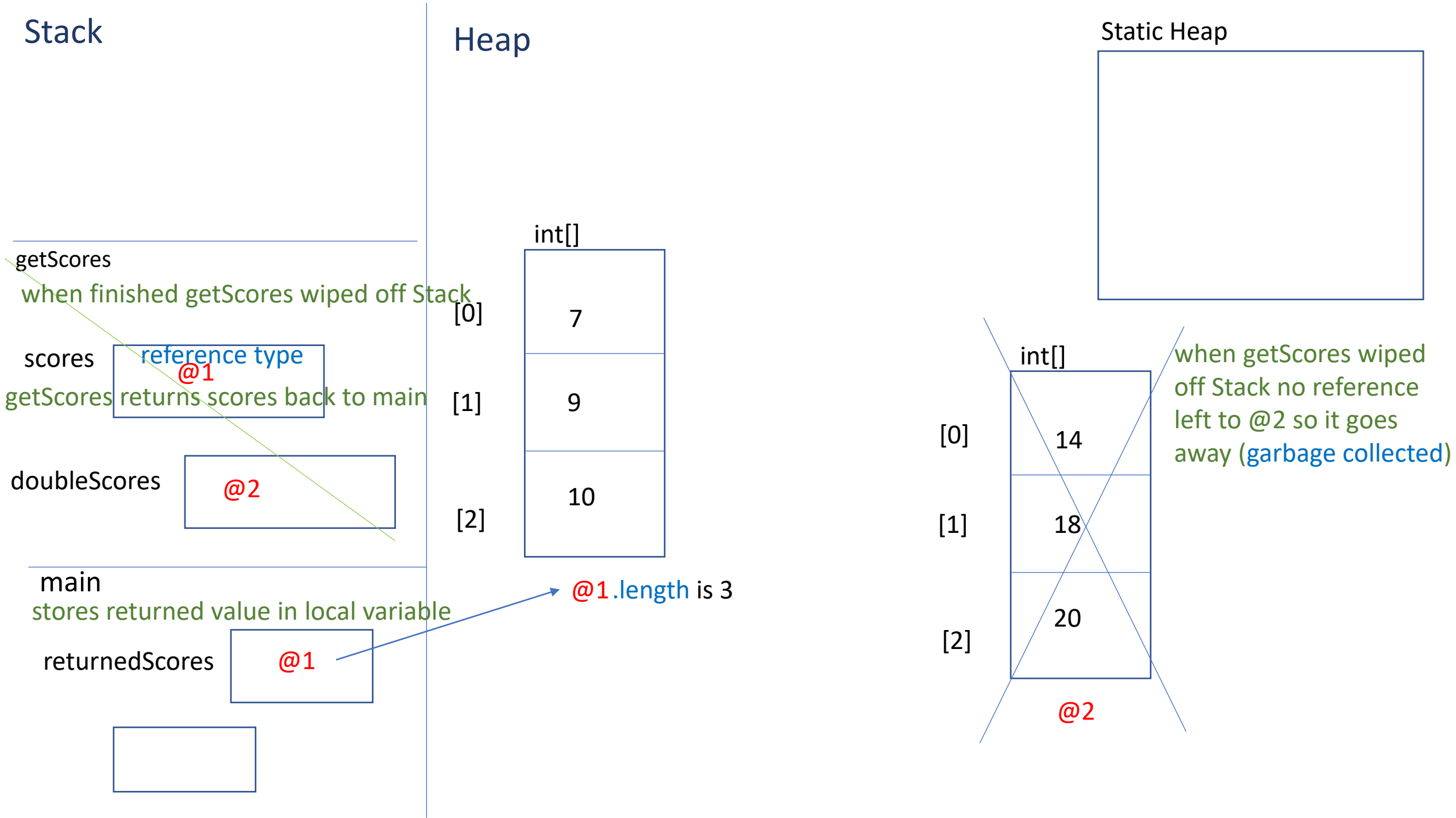
block for concrete method

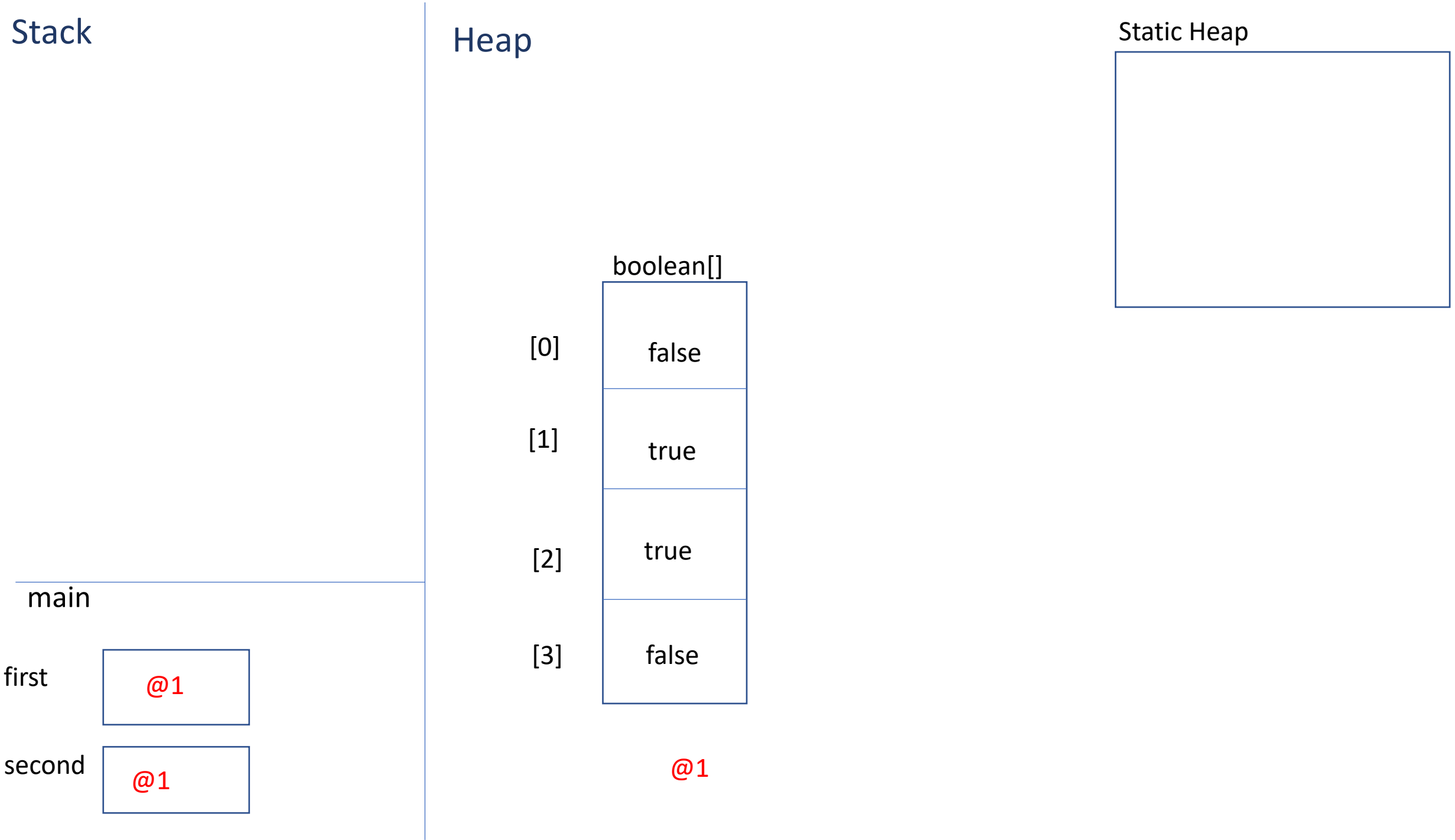
modifiers (others include static and final)

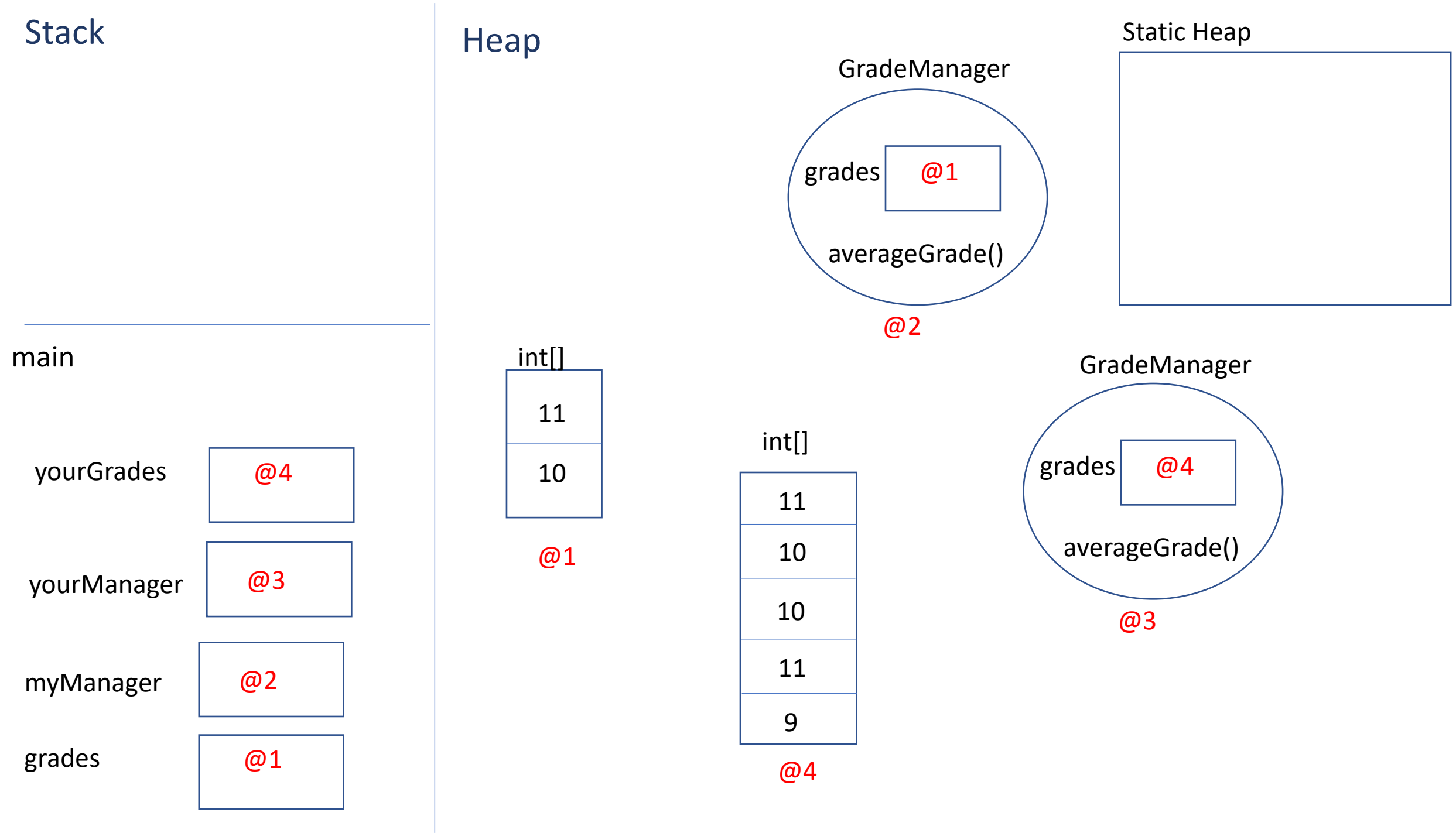
scope
- private – class itself
- default (no modifier) – plus other classes in same package
- protected – plus subclasses in another package
- public – plus all other classes

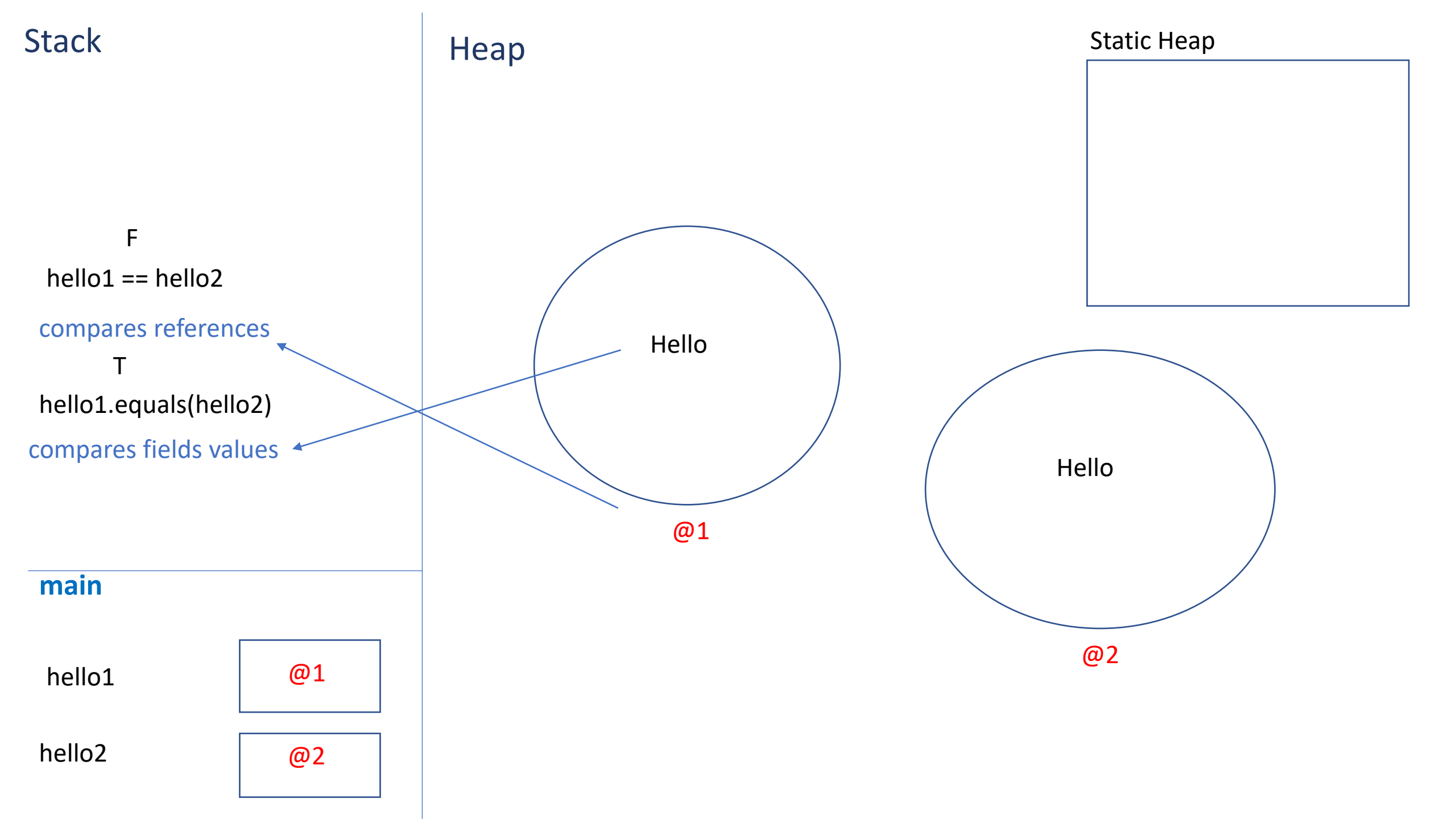

Stack
Heap
Static Heap
String class methods are ***immutable***
- does not change original String object
- creates a new String object with the changed value and returns the reference
original String object
new String object
david
DAVID
@1
@2
main
name
@1
upperName
@2

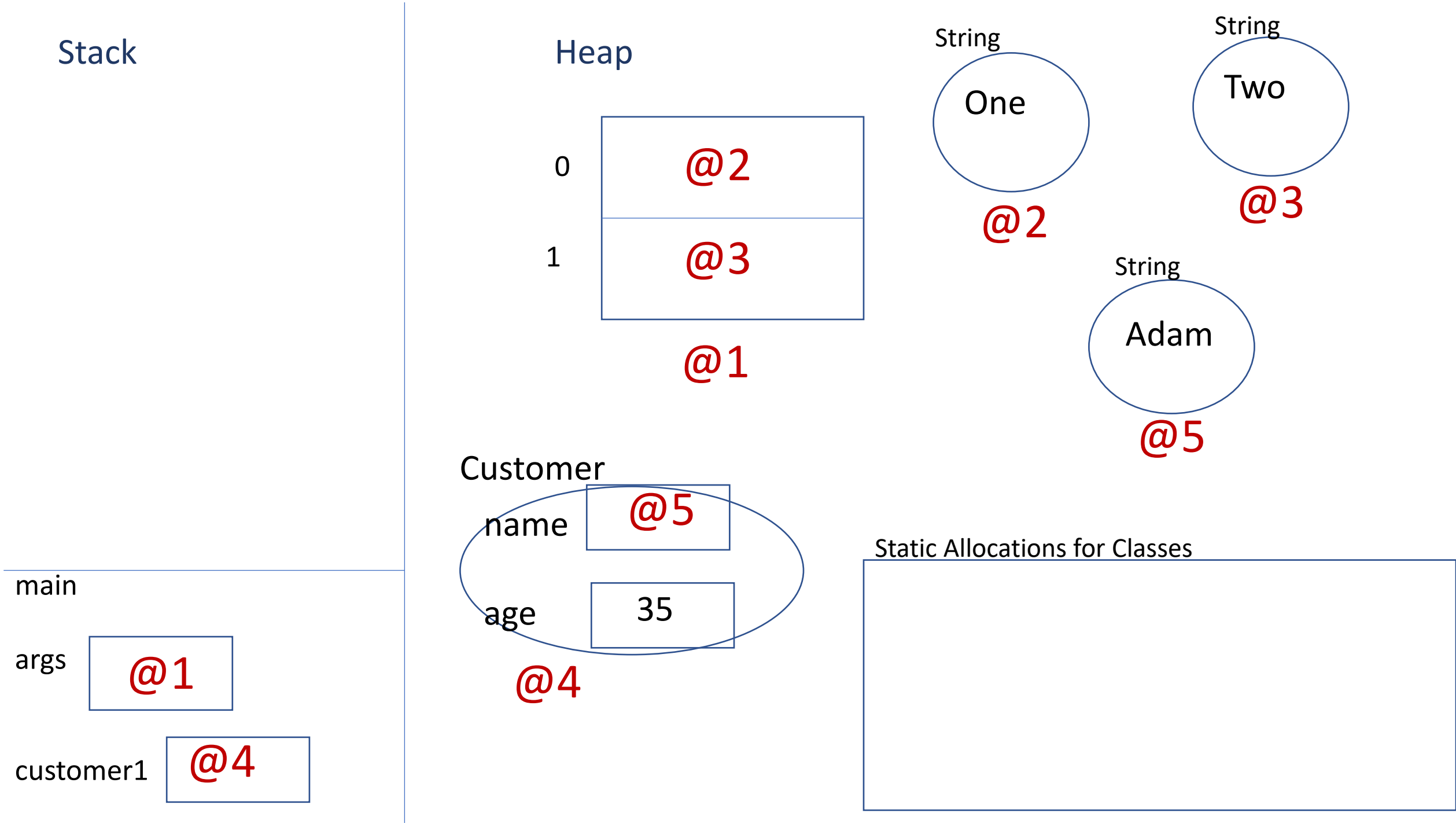

Stack
Heap
Static Heap
originally assigned a copy from num with value 99
isOkay
false
aBigNumber
52
return
99 over written when 52 assigned to aBigNumber
main
num
99
returnValue
false
notice when the Stack space for isOkay cleared upon return the value back in main for num remains the same (only a copy of the value gets passed into the parameter isBigNumber)…
changes to aBigNumber do NOT change num

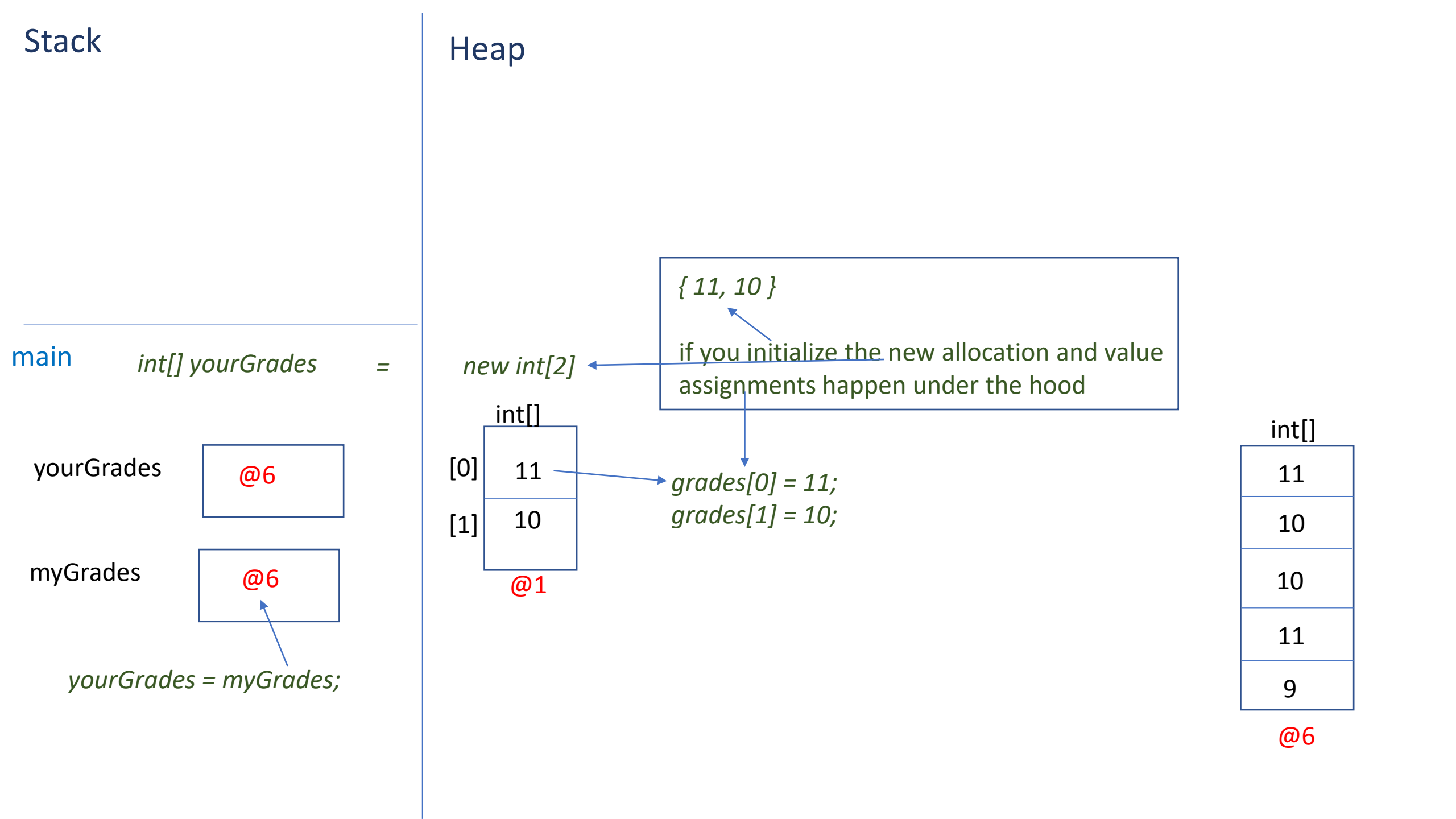

Stack
Heap
Static Heap
getScores
when finished getScores wiped off Stack
scores
reference type
@1
getScores returns scores back to main
doubleScores
@2
main
stores returned value in local variable
returnedScores
@1
int[]
[0]
7
[1]
9
[2]
10
@1.length is 3
int[]
[0]
14
[1]
18
[2]
20
@2
when getScores wiped off Stack no reference left to @2 so it goes away (garbage collected)

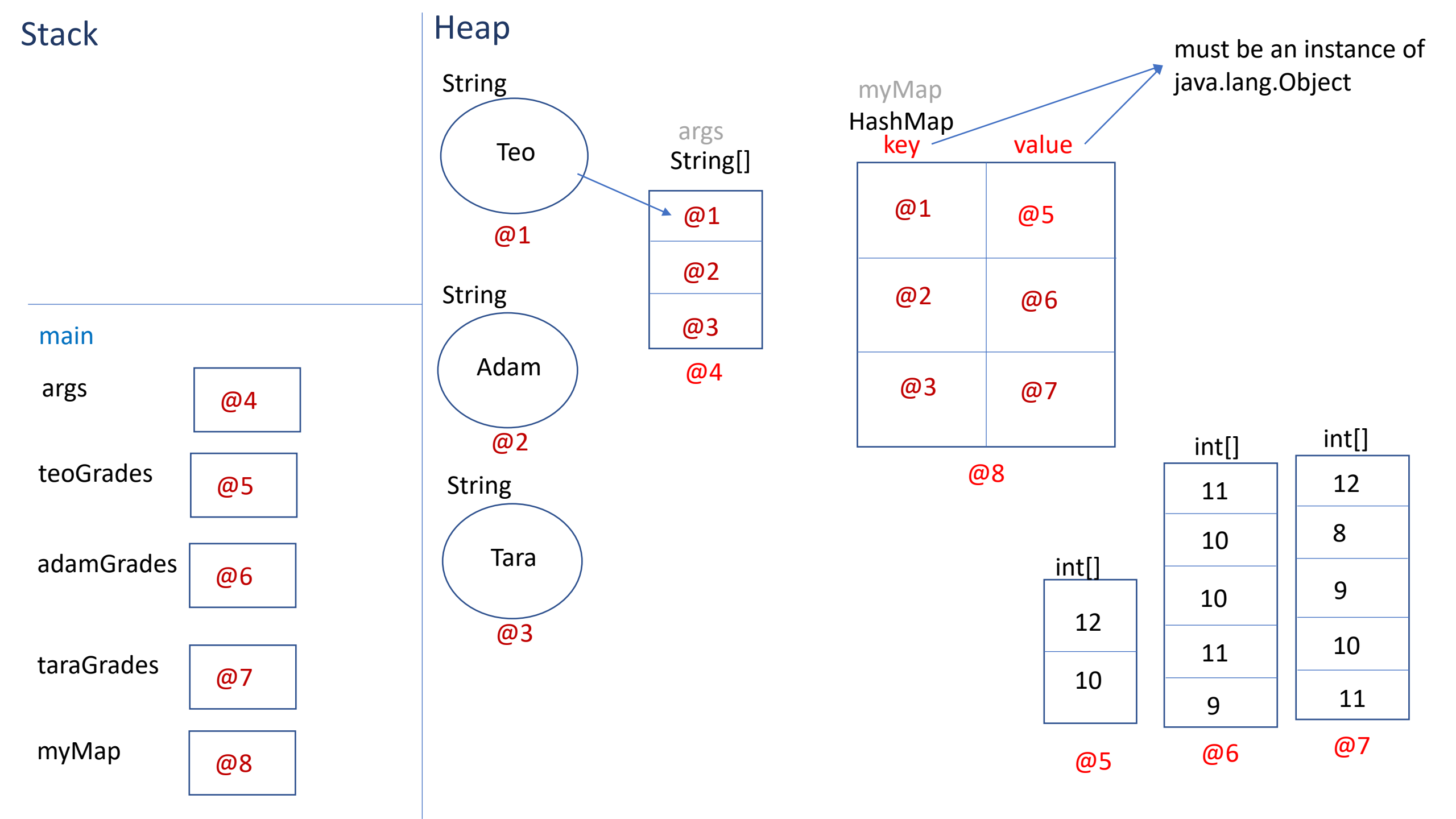

Stack
Heap
Static Heap
boolean[]
[0]
false
[1]
true
[2]
true
[3]
false
@1
main
first
@1
second
@1

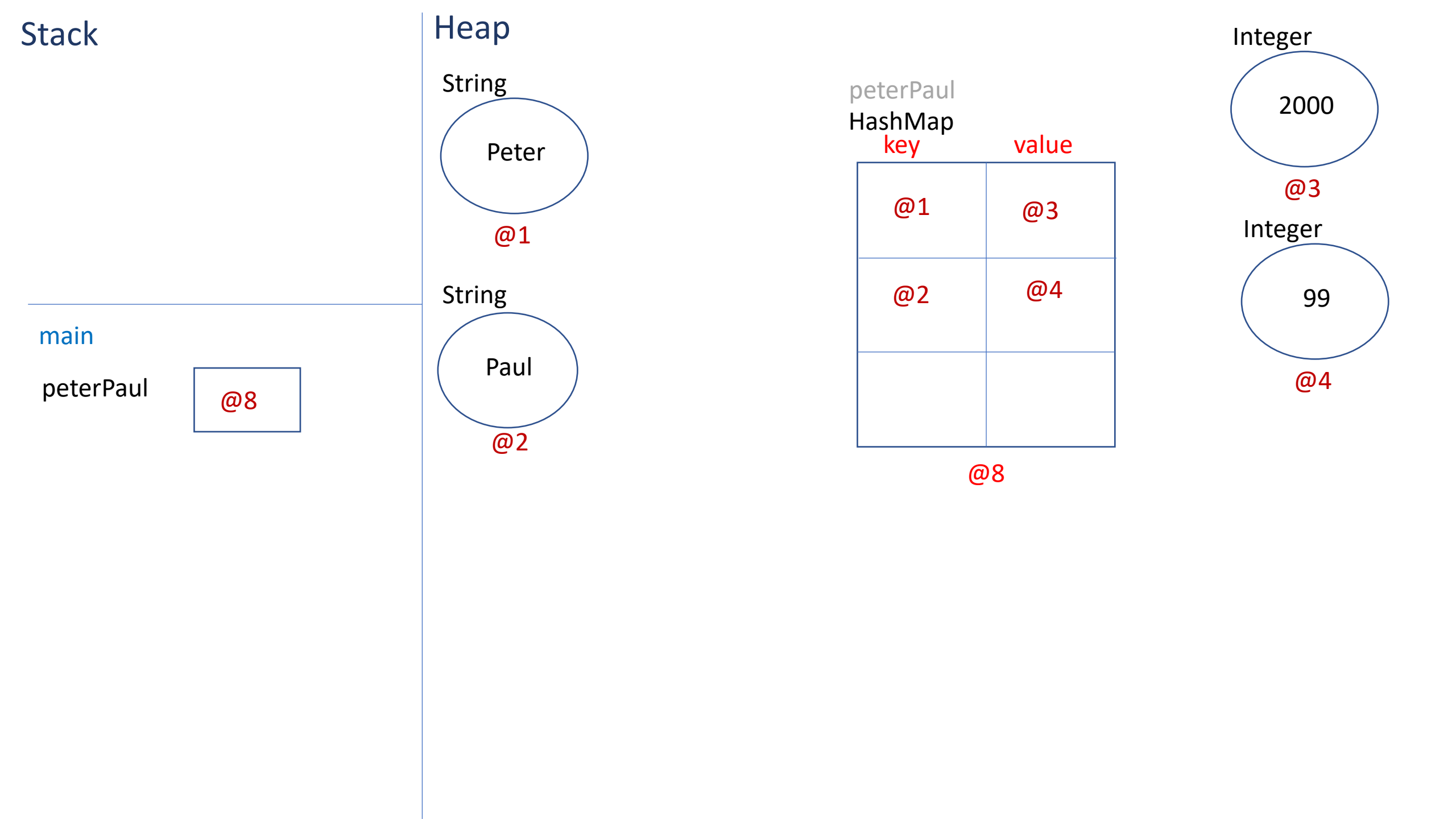

Stack
Heap
Static Heap
GradeManager
grades
@1
averageGrade()
@2
main
yourGrades
@4
yourManager
@3
myManager
@2
grades
@1
int[]
11
10
@1
int[]
11
10
10
11
9
@4
GradeManager
grades
@4
averageGrade()
@3

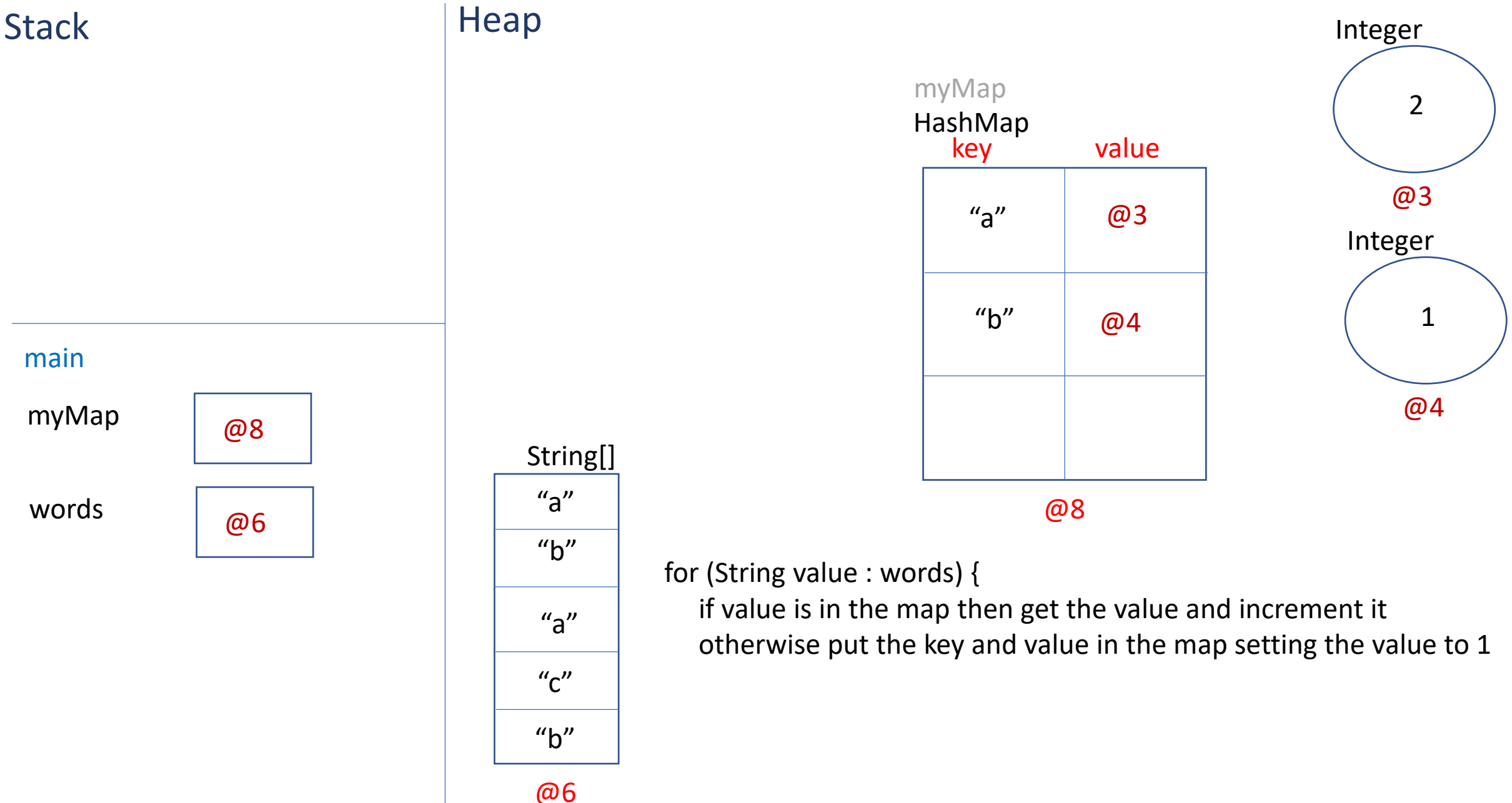

Stack
Heap
Static Heap
F
hello1 == hello2
compares references
T
hello1.equals(hello2)
compares fields values
Hello
@1
Hello
@2
main
hello1
@1
hello2
@2


Stack
Heap
0
@2
1
@3
@1
String
One
@2
String
Two
@3
String
Adam
@5
Customer
name
@5
age
35
@4
main
args
@1
customer1
@4
Static Allocations for Classes


Stack
Heap
main
int[] yourGrades =
yourGrades
@6
myGrades
@6
yourGrades = myGrades;
new int[2]
{ 11, 10 }
if you initialize the new allocation and value assignments happen under the hood
int[]
[0]
11
[1]
10
@1
grades[0] = 11;
grades[1] = 10;
int[]
11
10
10
11
9
@6


Stack
Heap
main
args
@4
teoGrades
@5
adamGrades
@6
taraGrades
@7
myMap
@8
String
Teo
@1
String
Adam
@2
String
Tara
@3
args
String[]
@1
@2
@3
@4
myMap
HashMap
key
value
@1
@5
@2
@6
@3
@7
@8
must be an instance of
java.lang.Object
int[]
12
10
@5
int[]
11
10
10
11
9
@6
int[]
12
8
9
10
11
@7


Stack
Heap
main
peterPaul
@8
String
Peter
@1
String
Paul
@2
peterPaul
HashMap
key
value
@1
@3
@2
@4
@8
Integer
2000
@3
Integer
99
@4




```
for (String value : words) {
    if value is in the map then get the value and increment it
    otherwise put the key and value in the map setting the value to 1
```

Application Architecture

Class – 1 general job to do – Single Responsibility Principal

Customer and Transaction classes – job is to transport the data needed across the tiers

lots of fields and setter/getters

Client

Driver class – job is user interaction

GUI or Console Application

Browser

HTML
JavaScript
CSS

Mobile

HTTP(s)
request

response

Web Integration

Web Server

Service
REST
SOAP

Business Rules

TransactionManager class – job is user business processing.

- needs fields Customer and Transaction
- methods for processTransaction() and shipOrder()

Resource Integration

CustomerDao class – job is to insert, update, delete, and query the database for Customers

TransactionDao class – job is to insert, update, delete, and query the database for Transactions.

Resources

File System

RDMS

Directory Services

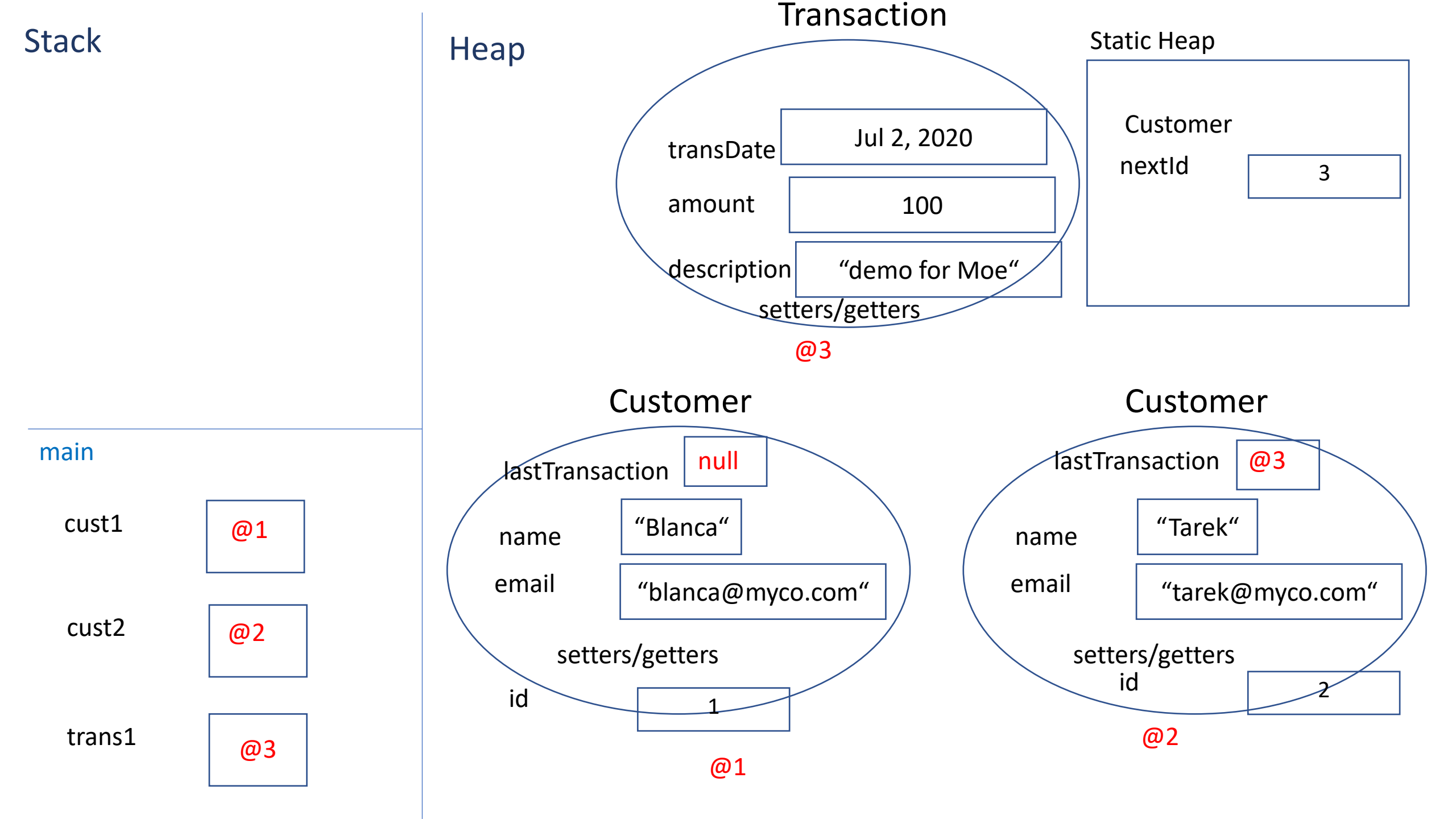
Message Services

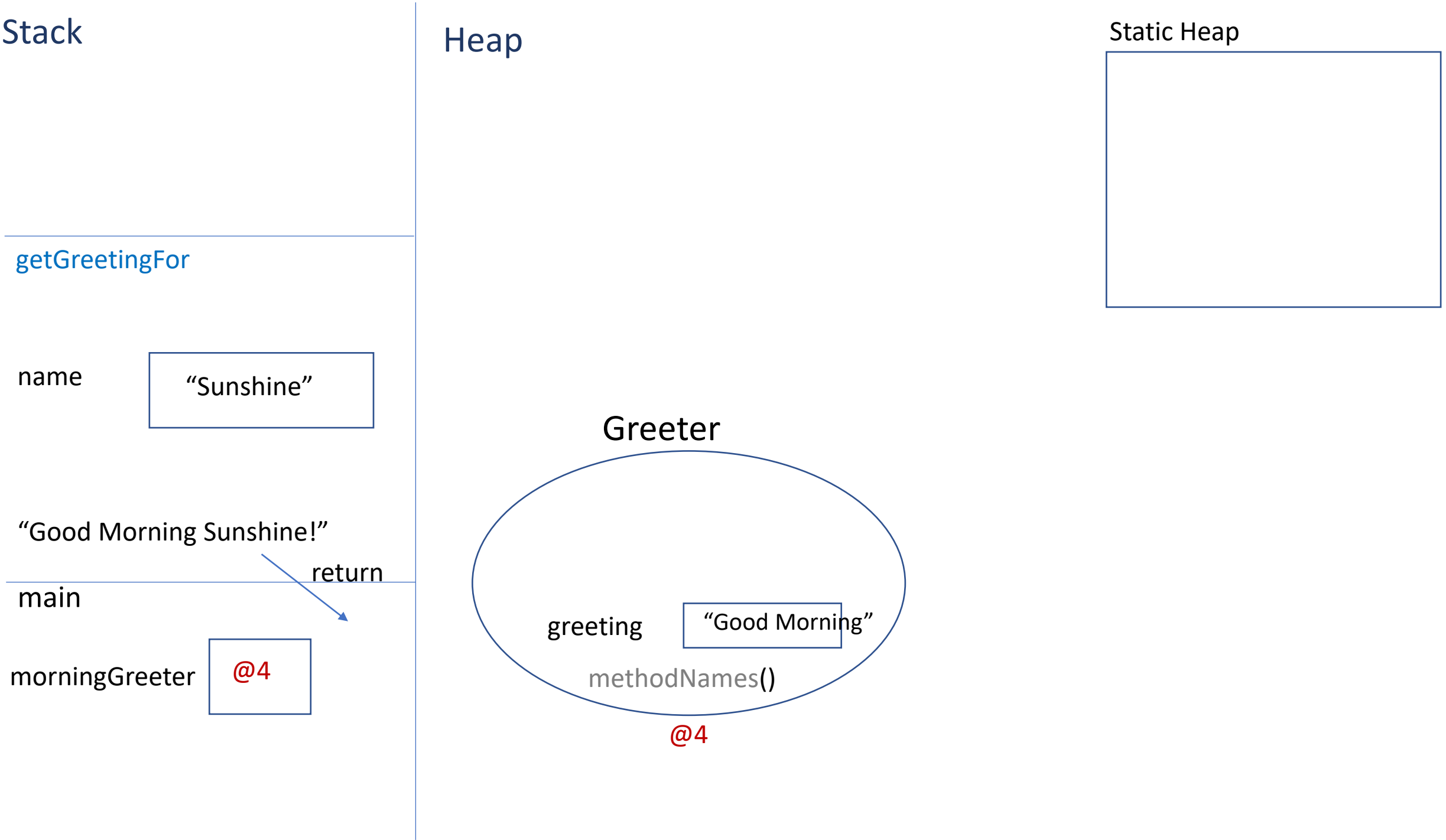
NonSQL database

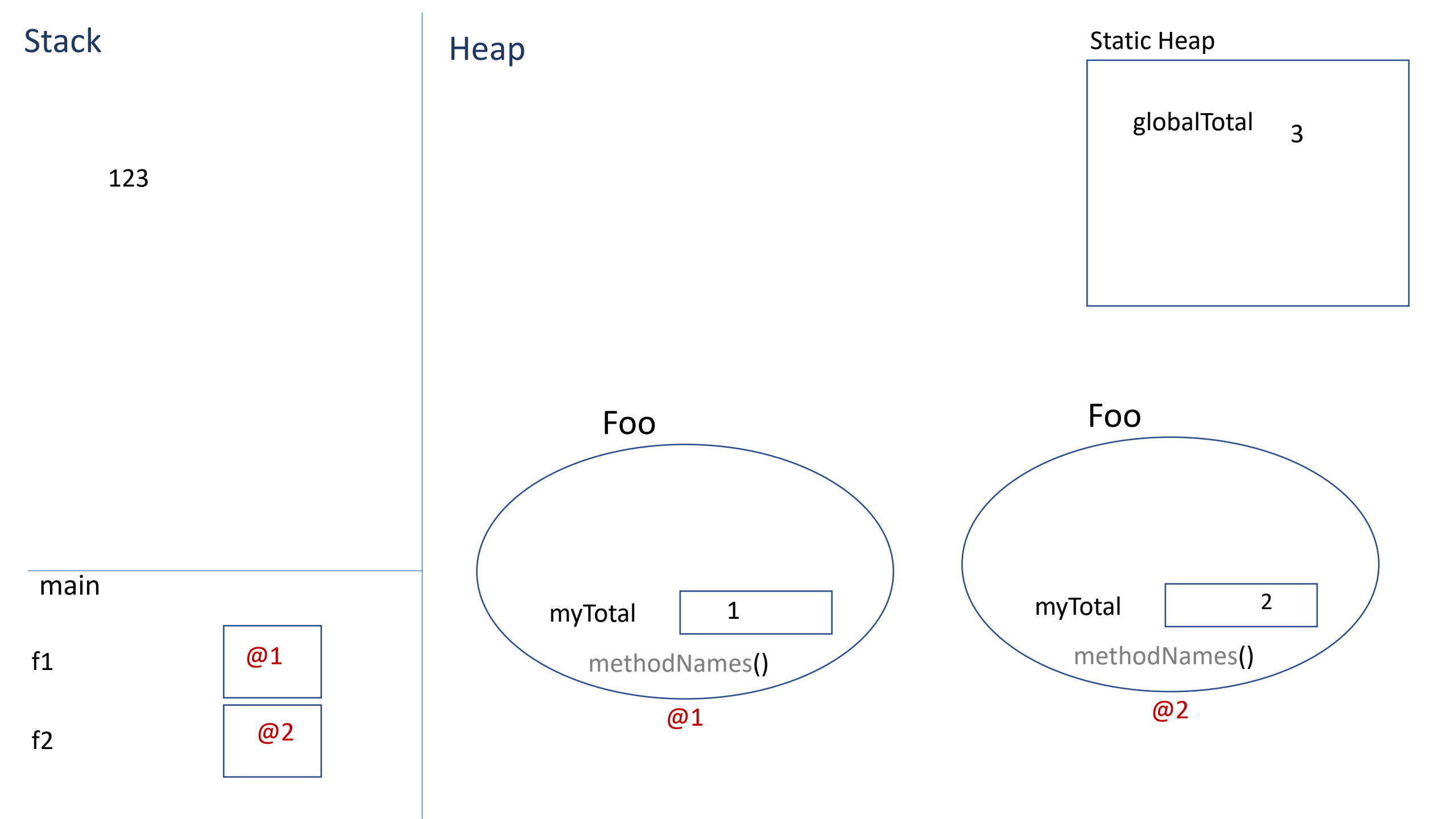
Other resources

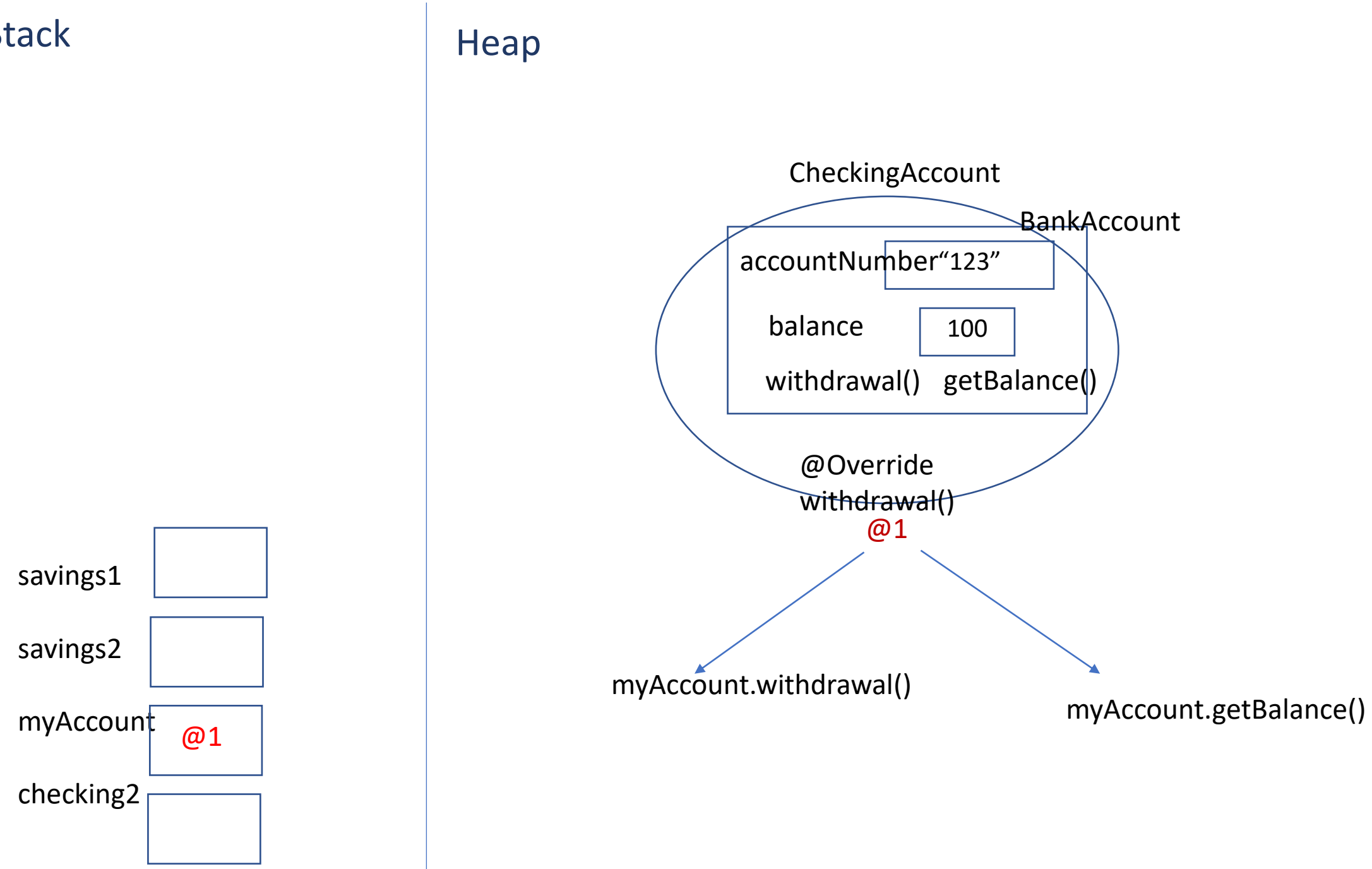
Security

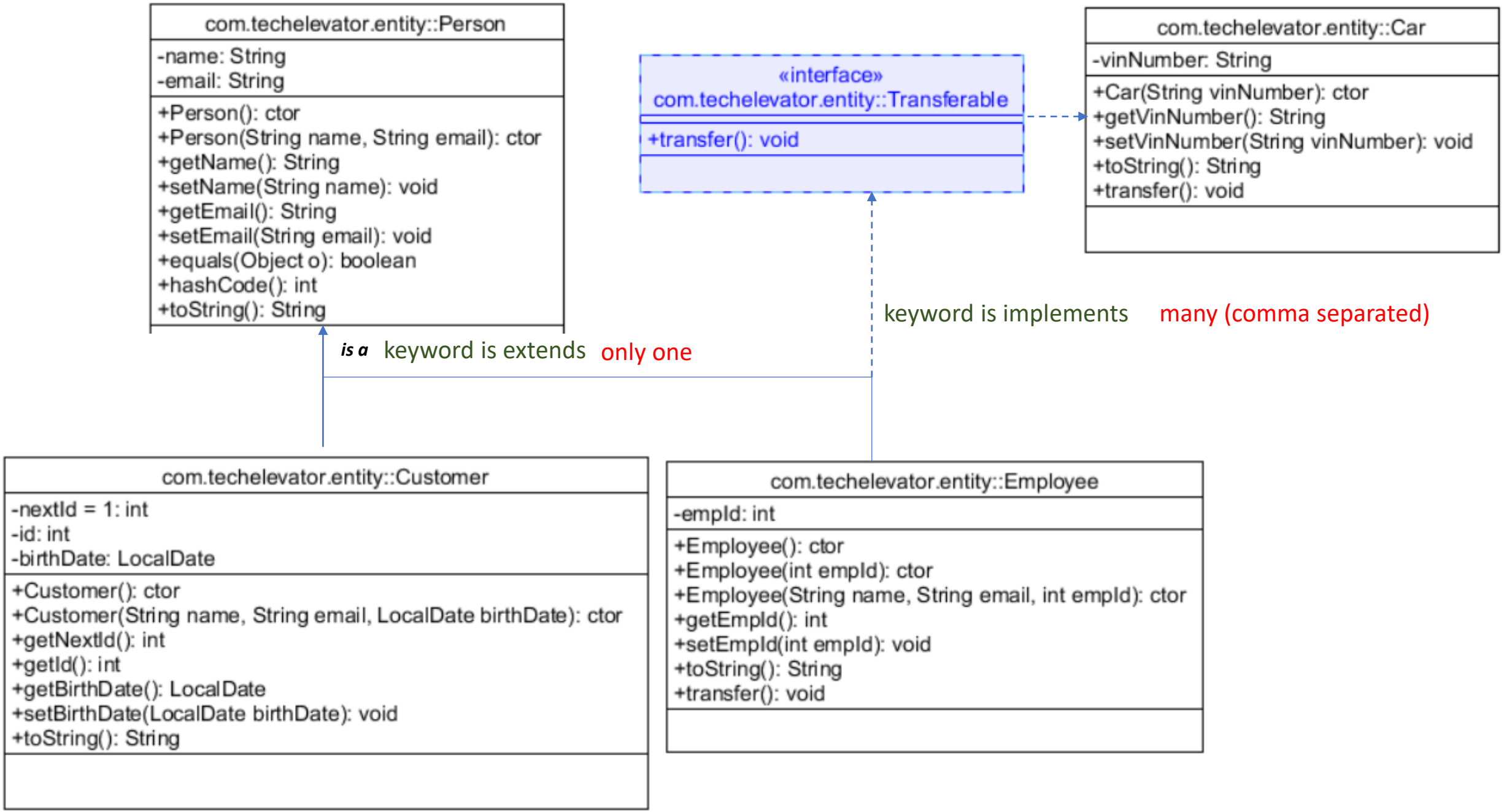
Authentication, Authorization, Encryption, CORS, Injection, …

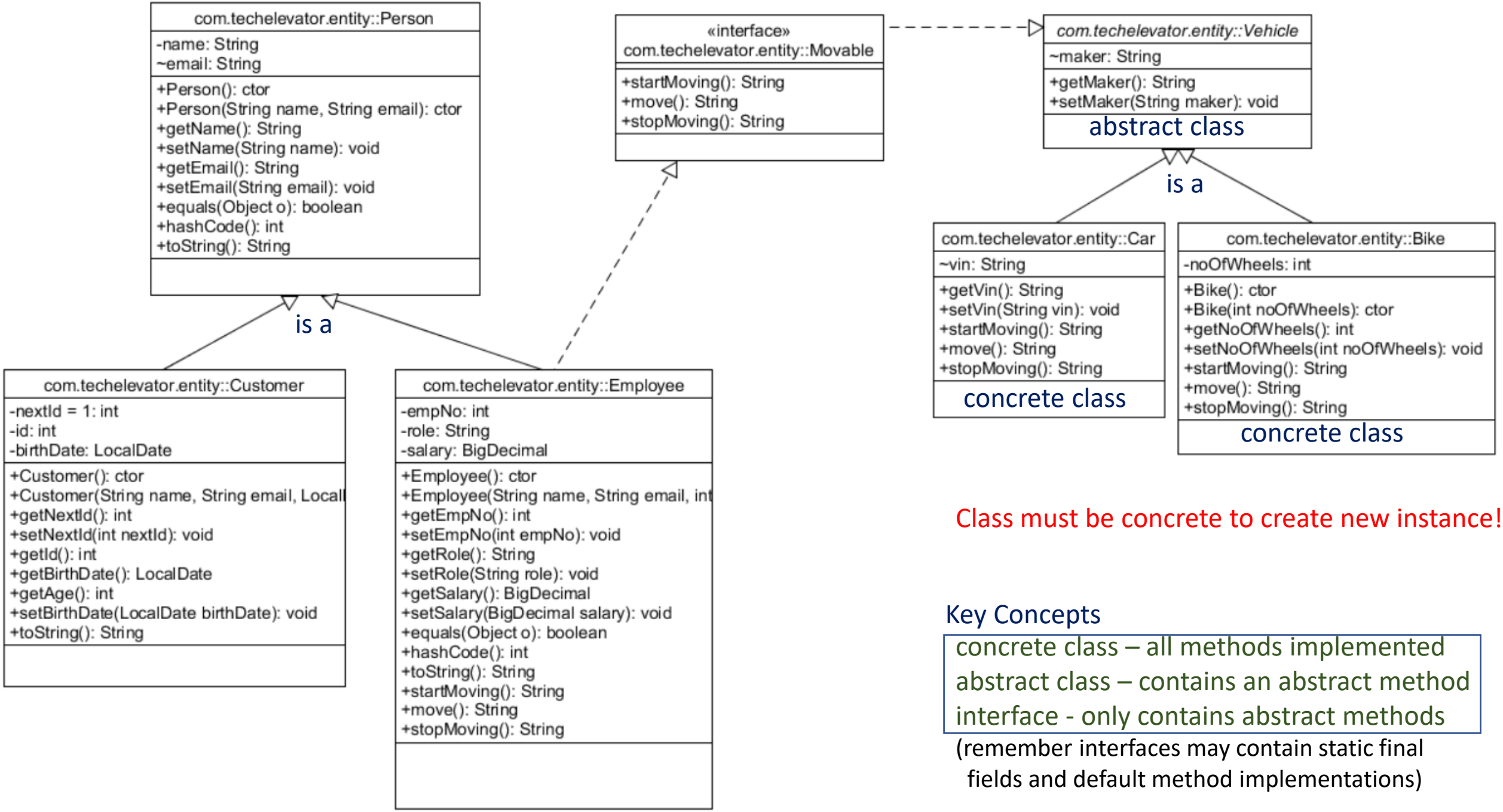

Stack
Heap
Transaction
transDate
Jul 2, 2020
amount
100
description
“demo for Moe“
setters/getters
@3
Static Heap
Customer
nextId
3
main
cust1
@1
cust2
@2
trans1
@3
Customer
lastTransaction
null
name
“Blanca“
email
“blanca@myco.com“
setters/getters
id
1
@1
Customer
lastTransaction
@3
name
“Tarek“
email
“tarek@myco.com“
setters/getters
id
2
@2


Stack
Heap
Static Heap
getGreetingFor
name
"Sunshine"
"Good Morning Sunshine!"
return
main
morningGreeter
@4
Greeter
greeting
"Good Morning"
methodNames()
@4


Stack
123
main
f1
@1
f2
@2
Heap
Static Heap
globalTotal
3
Foo
myTotal
1
methodNames()
@1
Foo
myTotal
2
methodNames()
@2


Stack
Heap
savings1
savings2
myAccount
@1
checking2
CheckingAccount
BankAccount
accountNumber“123”
balance
100
withdrawal()
getBalance()
@Override
withdrawal()
@1
myAccount.withdrawal()
myAccount.getBalance()


com.techelevator.entity::Person
-name: String
-email: String
+Person(): ctor
+Person(String name, String email): ctor
+getName(): String
+setName(String name): void
+getEmail(): String
+setEmail(String email): void
+equals(Object o): boolean
+hashCode(): int
+toString(): String
«interface»
com.techelevator.entity::Transferable
+transfer(): void
com.techelevator.entity::Car
-vinNumber: String
+Car(String vinNumber): ctor
+getVinNumber(): String
+setVinNumber(String vinNumber): void
+toString(): String
+transfer(): void
keyword is implements
many (comma separated)
is a
keyword is extends
only one
com.techelevator.entity::Customer
-nextId = 1: int
-id: int
-birthDate: LocalDate
+Customer(): ctor
+Customer(String name, String email, LocalDate birthDate): ctor
+getNextId(): int
+getId(): int
+getBirthDate(): LocalDate
+setBirthDate(LocalDate birthDate): void
+toString(): String
com.techelevator.entity::Employee
-empId: int
+Employee(): ctor
+Employee(int empId): ctor
+Employee(String name, String email, int empId): ctor
+getEmpId(): int
+setEmpId(int empId): void
+toString(): String
+transfer(): void

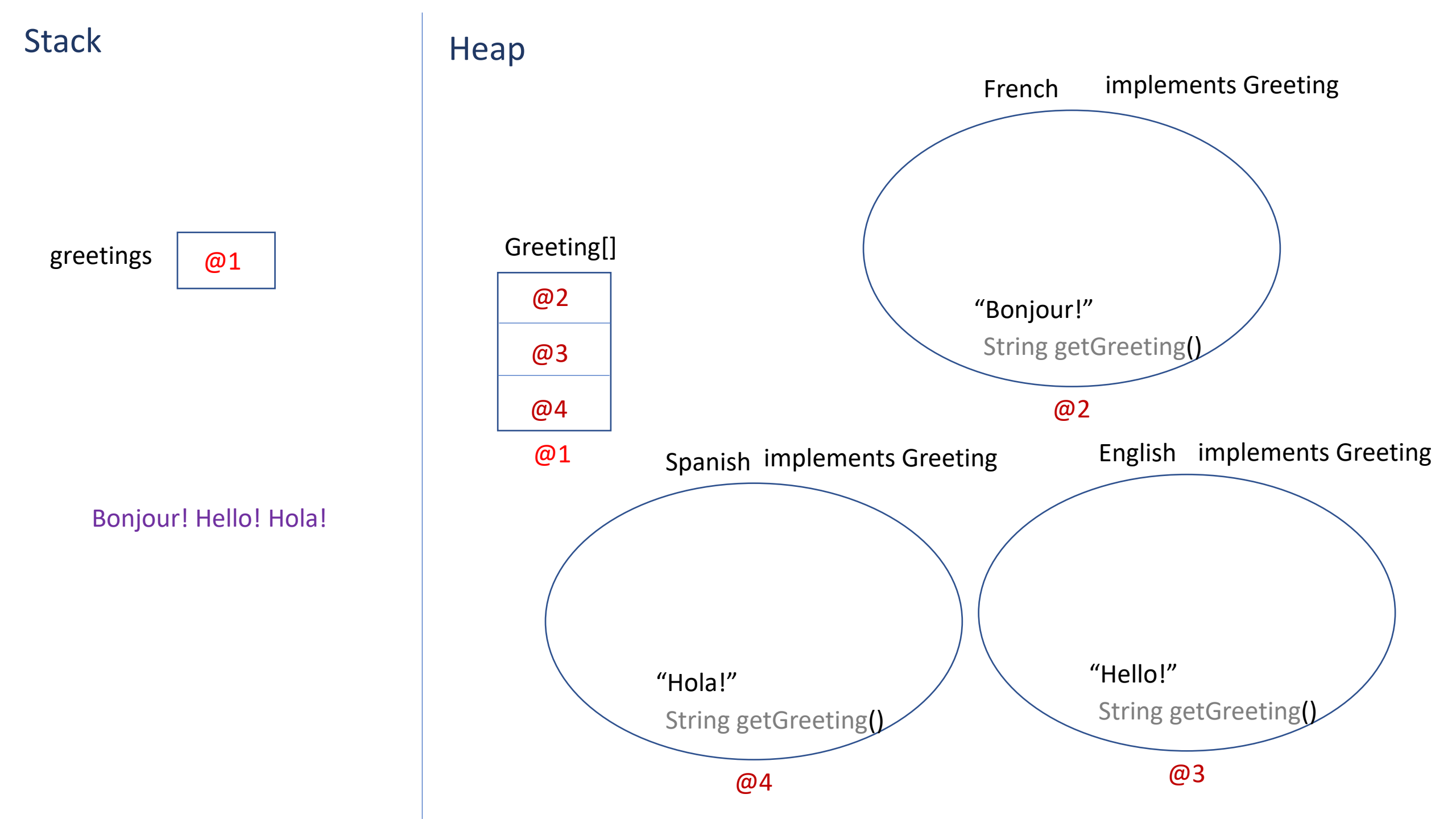


Class must be concrete to create new instance!

Key Concepts

concrete class – all methods implemented
abstract class – contains an abstract method
interface - only contains abstract methods

(remember interfaces may contain static final fields and default method implementations)


Stack
Heap
greetings
@1
Greeting[]
@2
@3
@4
@1
French
implements Greeting
"Bonjour!"
String getGreeting()
@2
Spanish
implements Greeting
"Hola!"
String getGreeting()
@4
English
implements Greeting
"Hello!"
String getGreeting()
@3
Bonjour! Hello! Hola!

Scope Modifiers

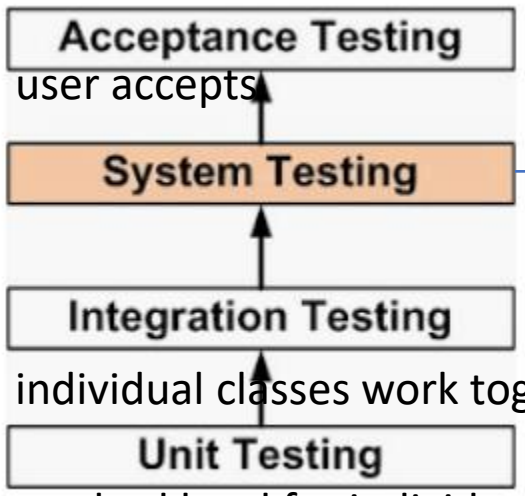
1. Default - classes in the same package
2. Private - only the class itself
3. Protected - subclasses get access
4. Public - access from any class

final on class – no subclass allowed

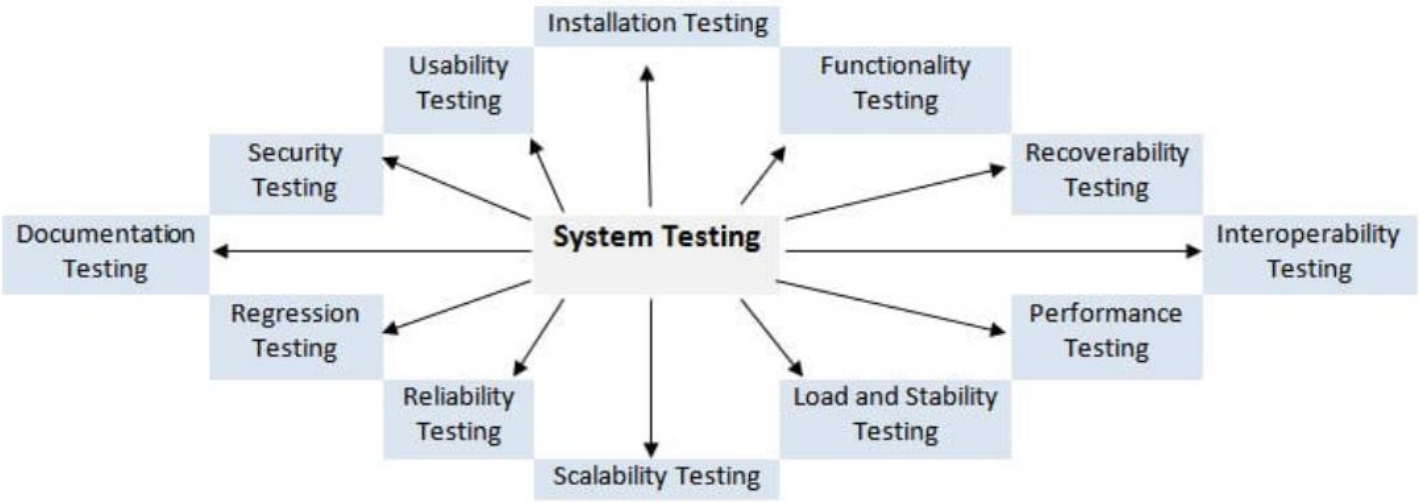
final on method – no override allowed

final on field – constant value

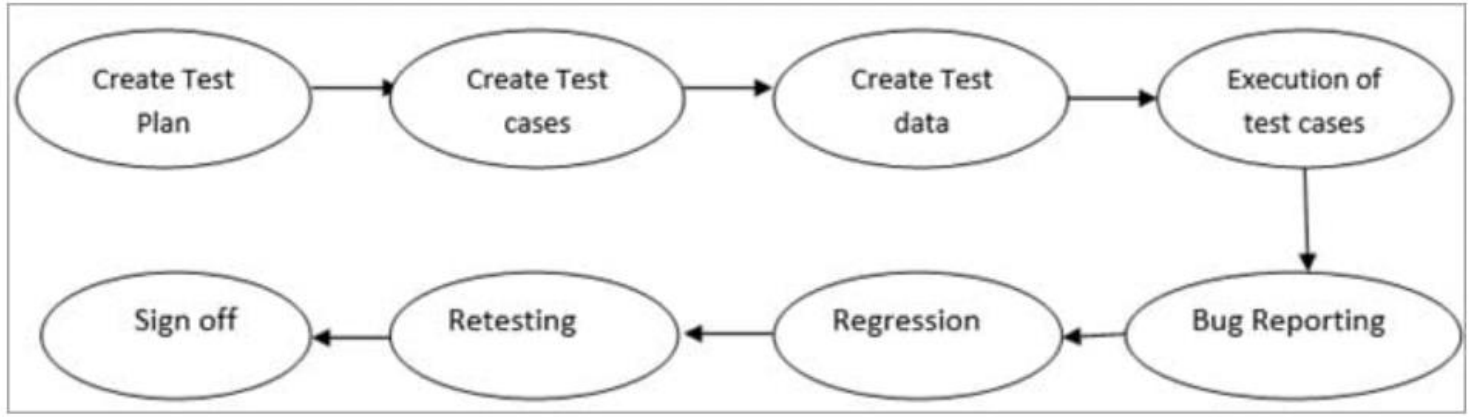
	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

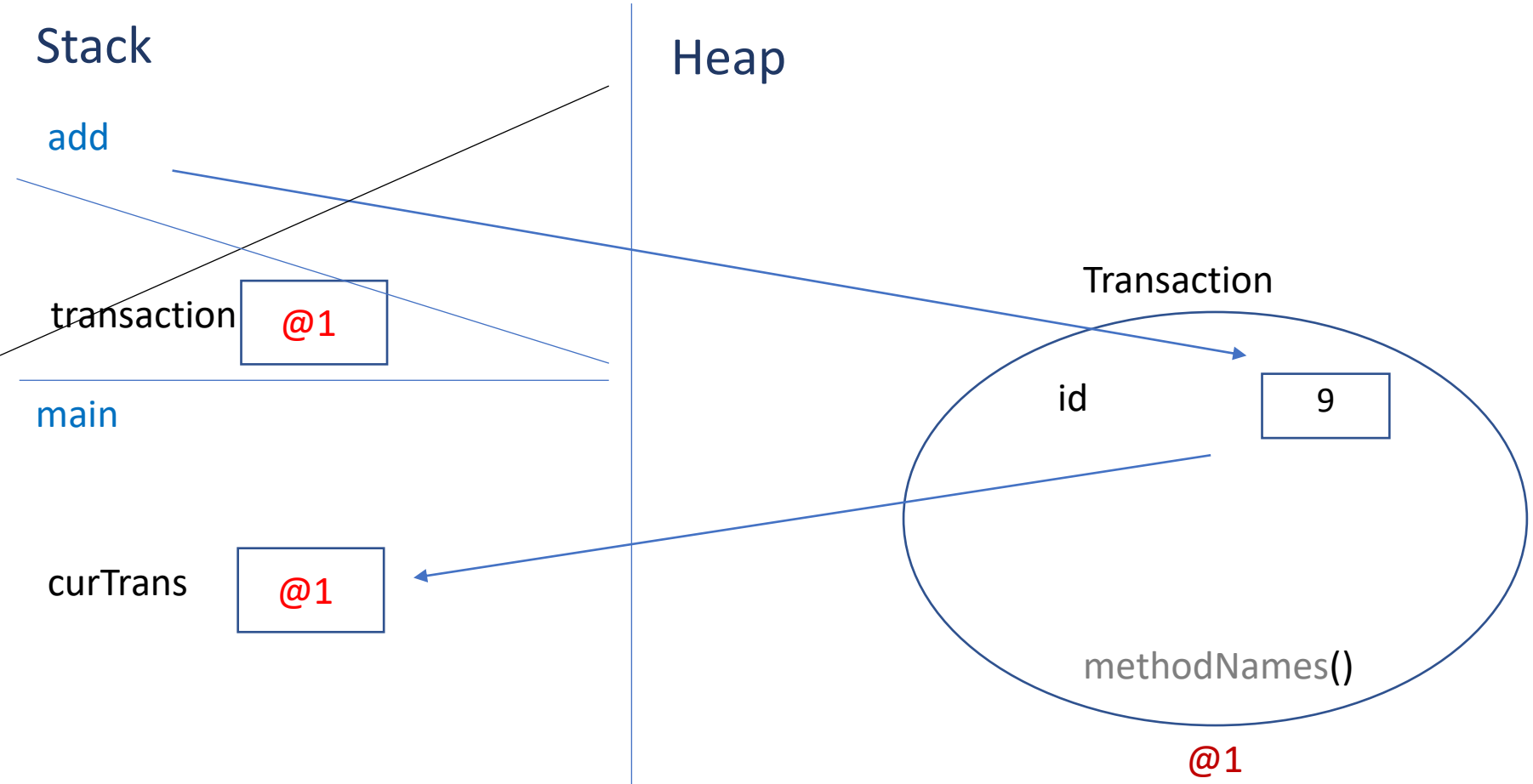


Types



Process




Stack
Heap
add
transaction
@1
main
curTrans
@1
Transaction
id
9
methodNames()
@1

Application Architecture

Client

Console Application

VendingMachineCLI

private VendingManager manager;

In main:
Logger.openLog(); // Logger.closeLog() on System.exit()

VendingManager =
  new VendingManager(InventoryDao.loadInventory());

In menu:
Use manager to **call** methods in Business tier
- feedMoney(amount)
- purchaseItem(slot)

Business Rules

VendingManager

Map<String, Item>
  vendingMachine;

Customer customer;

Business methods
implemented:
- feedMoney(amount)
- purchaseItem(slot)
…

Resource Integration

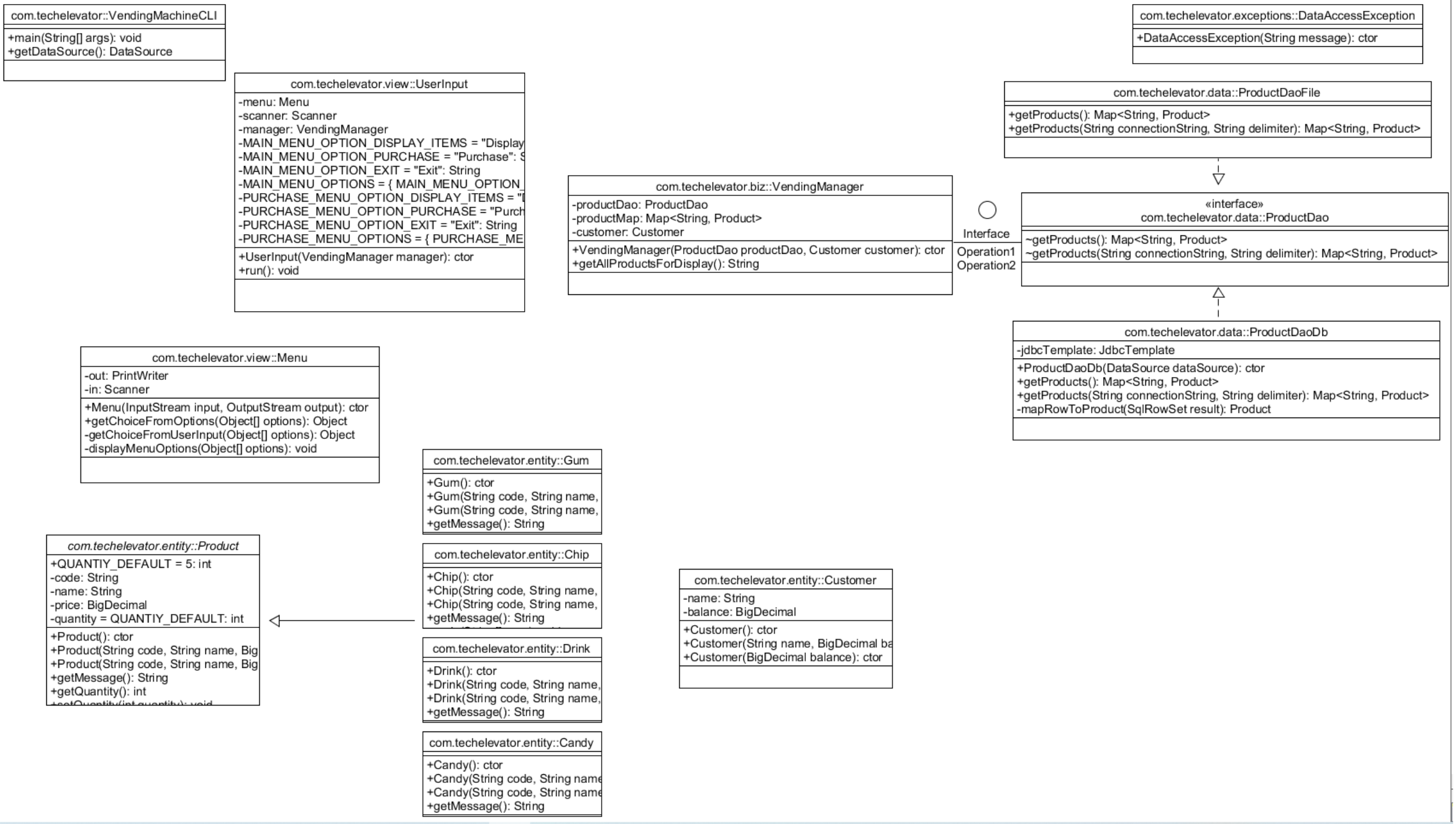
InventoryDao
- Read file
- Create map
- Initialize map into VendingMachine and return

Logger (all static)
- open()
- writeMessage()
- close()

Resources

File System

inventory.csv


com.techelevator::VendingMachineCLI
+main(String[] args): void
+getDataSource(): DataSource
com.techelevator.exceptions::DataAccessException
+DataAccessException(String message): ctor
com.techelevator.view::UserInput
-menu: Menu
-scanner: Scanner
-manager: VendingManager
-MAIN_MENU_OPTION_DISPLAY_ITEMS = "Display
-MAIN_MENU_OPTION_PURCHASE = "Purchase": S
-MAIN_MENU_OPTION_EXIT = "Exit": String
-MAIN_MENU_OPTIONS = { MAIN_MENU_OPTION_
-PURCHASE_MENU_OPTION_DISPLAY_ITEMS = "I
-PURCHASE_MENU_OPTION_PURCHASE = "Purch
-PURCHASE_MENU_OPTION_EXIT = "Exit": String
-PURCHASE_MENU_OPTIONS = { PURCHASE_ME
+UserInput(VendingManager manager): ctor
+run(): void
com.techelevator.data::ProductDaoFile
+getProducts(): Map<String, Product>
+getProducts(String connectionString, String delimiter): Map<String, Product>
com.techelevator.biz::VendingManager
-productDao: ProductDao
-productMap: Map<String, Product>
-customer: Customer
+VendingManager(ProductDao productDao, Customer customer): ctor
+getAllProductsForDisplay(): String
Interface
Operation1
Operation2
«interface»
com.techelevator.data::ProductDao
~getProducts(): Map<String, Product>
~getProducts(String connectionString, String delimiter): Map<String, Product>
com.techelevator.data::ProductDaoDb
-jdbcTemplate: JdbcTemplate
+ProductDaoDb(DataSource dataSource): ctor
+getProducts(): Map<String, Product>
+getProducts(String connectionString, String delimiter): Map<String, Product>
-mapRowToProduct(SqlRowSet result): Product
com.techelevator.view::Menu
-out: PrintWriter
-in: Scanner
+Menu(InputStream input, OutputStream output): ctor
+getChoiceFromOptions(Object[] options): Object
-getChoiceFromUserInput(Object[] options): Object
-displayMenuOptions(Object[] options): void
com.techelevator.entity::Gum
+Gum(): ctor
+Gum(String code, String name,
+Gum(String code, String name,
+getMessage(): String
com.techelevator.entity::Product
+QUANTIY_DEFAULT = 5: int
-code: String
-name: String
-price: BigDecimal
-quantity = QUANTIY_DEFAULT: int
+Product(): ctor
+Product(String code, String name, Big
+Product(String code, String name, Big
+getMessage(): String
+getQuantity(): int
com.techelevator.entity::Chip
+Chip(): ctor
+Chip(String code, String name,
+Chip(String code, String name,
+getMessage(): String
com.techelevator.entity::Drink
+Drink(): ctor
+Drink(String code, String name,
+Drink(String code, String name,
+getMessage(): String
com.techelevator.entity::Customer
-name: String
-balance: BigDecimal
+Customer(): ctor
+Customer(String name, BigDecimal ba
+Customer(BigDecimal balance): ctor
com.techelevator.entity::Candy
+Candy(): ctor
+Candy(String code, String name
+Candy(String code, String name
+getMessage(): String

Stack

interface IProductType

String getMessage()

abstract

Heap

abstract Product implements IProductType

Chip extends Product

myMap

HashMap<String, Product>

key	value
A1	@1

@8

code "A1"

name "Potato Crisps"

price 3.05

Product

methodNames()

String getMessage("Yum "Yum Crunch")

Chip

@1

Setup Postgres Database

1. Open command prompt
2. > createdb –U postgres databasename
3. Open DbVisualizer
4. Right-click on Connections and select Create Database Connection
5. If prompted click Use Wizard button and follow Wizard
   - Select Database Driver – PostgresSQL
   - Database name – databasename from command line prompt above
   - Username – postgres
   - Password – postgres1
6. With new connection selected open and run database script file

Columns

Primary Key Column

Rows

Code	Name	Continent	Region
CYM	Cayman Islands	North America	Caribbean
CHL	Chile	South America	South America
COK	Cook Islands	Oceania	Polynesia
CRI	Costa Rica	North America	Central America
DJI	Djibouti	Africa	Eastern Africa
DMA	Dominica	North America	Caribbean
DOM	Dominican Republic	North America	Caribbean
ECU	Ecuador	South America	South America
EGY	Egypt	Africa	Northern Africa
SLV	El Salvador	North America	Central America
ERI	Eritrea	Africa	Eastern Africa
ESP	Spain	Europe	Southern Europe
ZAF	South Africa	Africa	Southern Africa
ETH	Ethiopia	Africa	Eastern Africa
FLK	Falkland Islands	South America	South America
FJI	Fiji Islands	Oceania	Melanesia
PHL	Philippines	Asia	Southeast Asia
FRO	Faroe Islands	Europe	Nordic Countries
GAB	Gabon	Africa	Central Africa

Country Table

```
SELECT col1, col2
   FROM tablename
   WHERE col1 = 'value'
   ORDER BY col1 [ASC | DESC], col2 [ASC | DESC]
```

```
SELECT expression1, expression2, ... expression_n,
      aggregate_function (aggregate_expression)
   FROM tables [where condition_expression]
   GROUP BY expression1, expression2, ... expression_n
   ORDER BY
```

*The **GROUP BY** clause can be used in conjunction with a SELECT statement and aggregate functions to collect data across multiple records.* It will return one record for each group.

```
SELECT column_name [, column_name]
   FROM table1 [, table2]
   WHERE column_name (IN | NOT IN)
      (SELECT column_name FROM table [WHERE])
```

*A **subquery** is referred to as an inner query and can provide the results of one query as input to another. Can only return **one** item in the SELECT clause.*

Aggregate Functions

AVG - returns the average value of a numeric column
SUM - returns the total sum of a numeric column
COUNT - returns the number of rows matching criteria
MIN - returns the smallest value of the selected column
MAX - returns the largest value of the selected column

Postgres Concatenate Strings

```
SELECT (column1 || ', ' || column2) FROM table
```

Keys - used to create relationships between two tables.

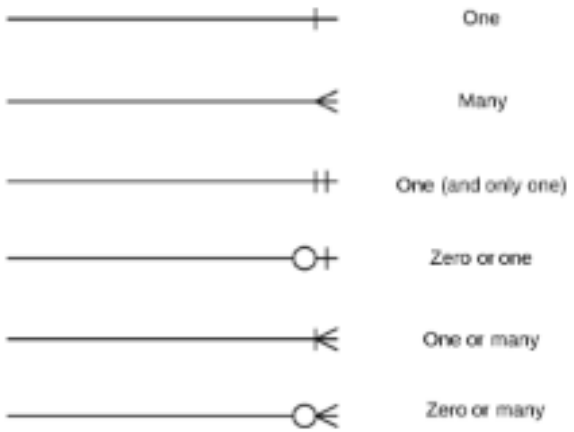
Types

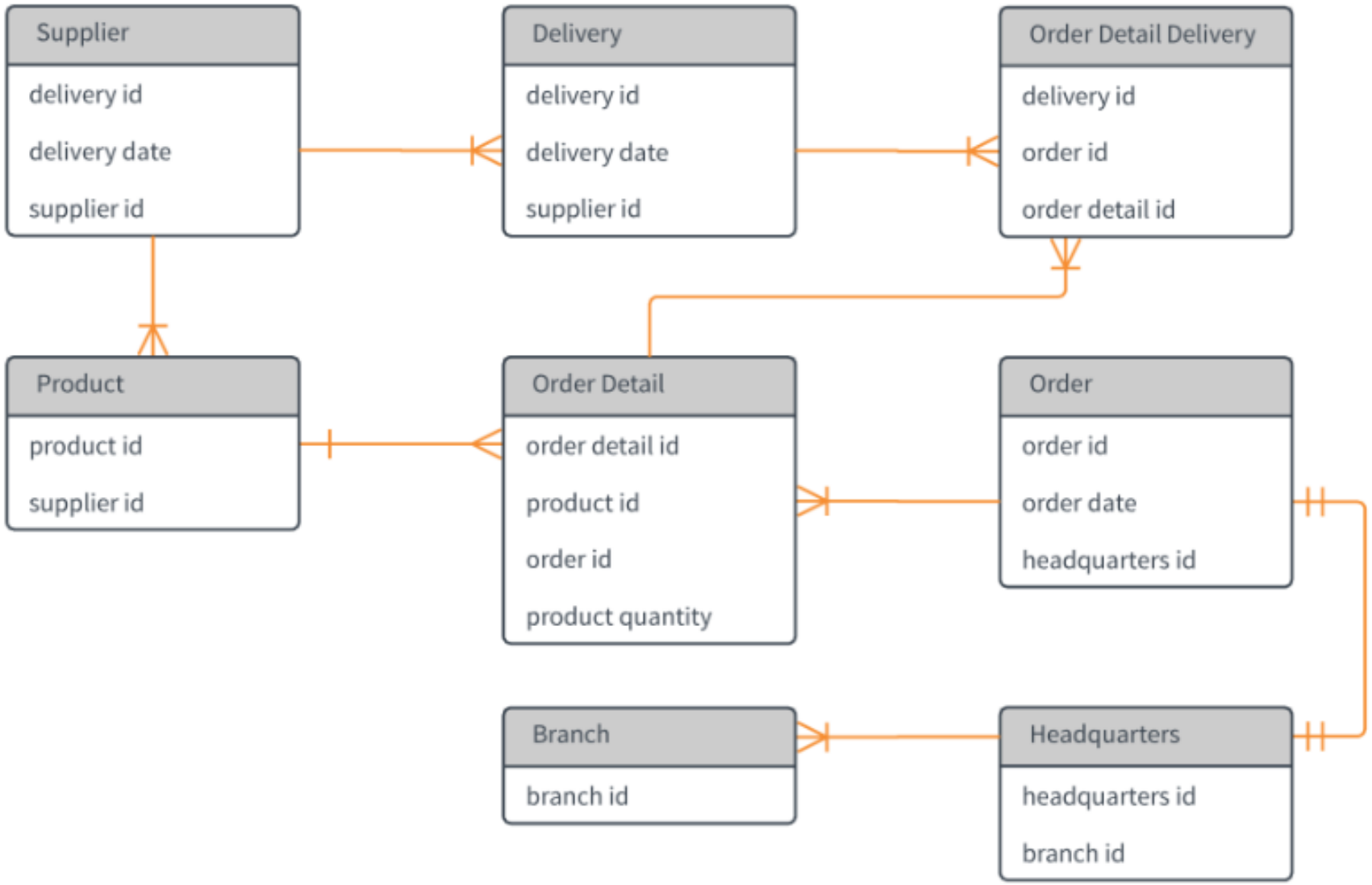
1. Natural Keys are formed from values in the real world (e.g. SSN, ISBN, Tax Id, E-mail?)
2. Surrogate Keys are artificially created by the application and identify a unique record.
3. Primary Keys uniquely identify each row within a table. They cannot be duplicated within that table and cannot be null. It is typically a single column but can also be comprised of multiple columns.
4. Foreign Keys exist in other tables and are used to reference a primary key in the source table.

Cardinality:

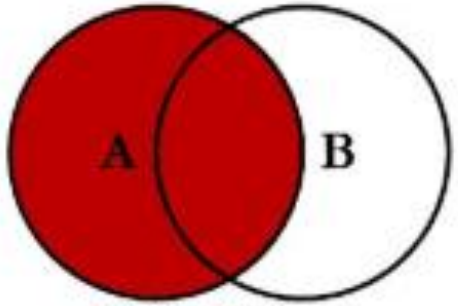
Refers to the maximum number of times that an instance in one entity can be associated with instances in a related entity, along with the minimum number of times it must be associated

Cardinality **Symbols**

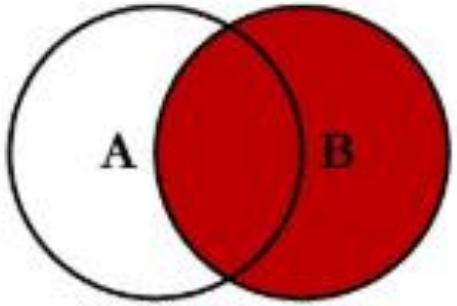



Supplier
delivery id
delivery date
supplier id
Delivery
delivery id
delivery date
supplier id
Order Detail Delivery
delivery id
order id
order detail id
Product
product id
supplier id
Order Detail
order detail id
product id
order id
product quantity
Order
order id
order date
headquarters id
Branch
branch id
Headquarters
headquarters id
branch id

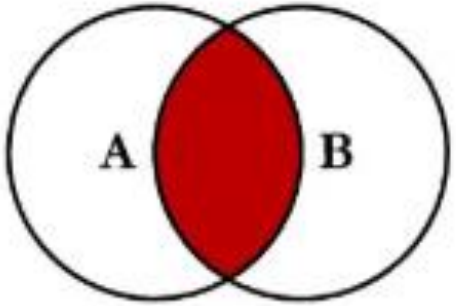
SQL JOINS



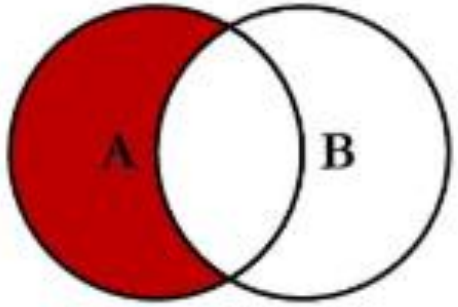

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



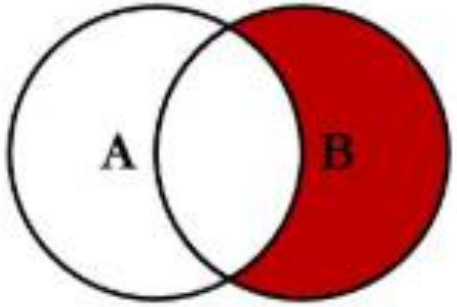

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



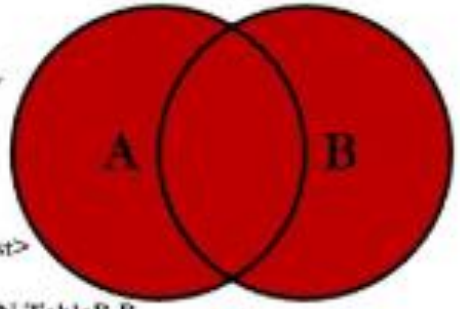

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



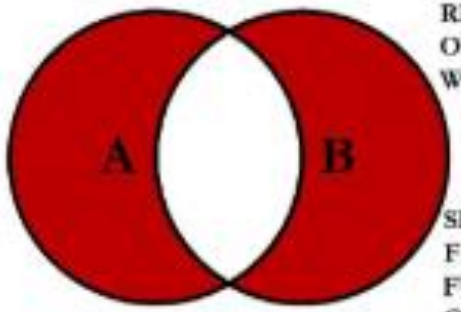

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

A **transaction** is an **atomic** (single) unit of work. If successful you **commit**, if error you **rollback**.

```
START TRANSACTION

COMMIT

ROLLBACK
```

```
INSERT INTO table_name (column1, column2, ..., column_n)
VALUES (value1, value2, ... value_n);

INSERT INTO countrylanguage (countrycode, language, isofficial, percentage)
VALUES ('USA', 'Klingon', false, 99);

UPDATE table_name
SET column = value
WHERE column = value;

UPDATE country
SET capital = 3796
WHERE code = 'USA';

DELETE FROM table_name
WHERE column=value;
```

Don't forget the WHERE clause!!!

```
DELETE FROM countrylanguage
WHERE countrycode = 'USA' AND language = 'English';
```

REFERENTIAL INTEGRITY

Referential integrity ensures that relationships between tables remain consistent!

Should I be allowed to delete a customer from the customers table if the customer's primary key exists as a foreign key in the transactions table?

We **enforce** referential integrity and other **rules** to keep our database tables and relationships correct with **constraints**.

A **constraint** is associated with a table and defines **properties** for which the column data must **comply**.

Types of Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK – specifies acceptable values that can be entered into the column
- DEFAULT – provides a default value for the column

Gallery Customer History Form

Customer Name

Jackson, Elizabeth Phone (206) 284-6783
123 – 4th Avenue
Fonthill, ON
L3J 4S4

Purchases Made

Artist	Title	Purchase Date	Sales Price
03 - Carol Channing	Laugh with Teeth	09/17/2000	7000.00
15 - Dennis Frings	South toward Emerald Sea	05/11/2000	1800.00
03 - Carol Channing	At the Movies	02/14/2002	5550.00
15 - Dennis Frings	South toward Emerald Sea	07/15/2003	2200.00

The Gill Art Gallery wishes to maintain data on their customers, artists and paintings. They may have several paintings by each artist in the gallery at one time. Paintings may be bought and sold several times. In other words, the gallery may sell a painting, then buy it back at a later date and sell it to another customer.

1st Normalized Form (NF) – every column contains only a single value.

No table should contain duplicative columns that one could use to get other types of information.

ID	Name	Start Year	Course Title
1	Arianna Kazemi	2020	Java Introduction
1	Arianna Kazemi	2020	SQL Database Access
1	Arianna Kazemi	2020	Spring MVC
2	Blanca Lopez	2020	Java Introduction
3	Cailey Farrell	2020	REST Mirco Services
4	Cesar Bueno	2020	SQL Database Access
5	Chris Amarvi	2020	REST Mirco Services

✓ **1st NF**

ID	First	Last	Start Year	Course Title
1	Arianna	Kazemi	2020	Java Introduction
1	Arianna	Kazemi	2020	SQL Database Access
1	Arianna	Kazemi	2020	Spring MVC
2	Blanca	Lopez	2020	Java Introduction
3	Cailey	Farrell	2020	REST Mirco Services
4	Cesar	Bueno	2020	SQL Database Access
5	Chris	Amarvi	2020	REST Mirco Services

2nd NF – all non-key columns must depend on the whole primary key (must be about the same thing)

Students

ID	First	Last	Start Year	Course Title	Credits	Description
1	Arianna	Kazemi	2020	Java Introduction	4	Learn Java
1	Arianna	Kazemi	2020	SQL Database Access	2	Write SQL
1	Arianna	Kazemi	2020	Spring MVC	2	Web development
2	Blanca	Lopez	2020	Java Introduction	4	Learn Java
3	Cailey	Farrell	2020	REST Mirco Services	3	Write services
4	Cesar	Bueno	2020	SQL Database Access	2	Write SQL
5	Chris	Amarvi	2020	REST Mirco Services	3	Write services

✓ **2nd NF**

Students

ID	First	Last	Start Year
1	Arianna	Kazemi	2020
2	Blanca	Lopez	2020
3	Cailey	Farrell	2020
4	Cesar	Bueno	2020
5	Chris	Amarvi	2020

Courses

ID	Course Title	Credits	Description
1	Java Introduction	4	Learn Java
2	SQL Database Access	2	Write SQL
3	Spring MVC	2	Web development
4	REST Mirco Services	3	Write services

Functional dependency:

First name column is functionally dependent on ID primary key

ID -> First

(there is only 1 first name for every student)

Every non-key column must be functionally dependent on the primary key.

Students

ID	First	Last	Start Year
1	Arianna	Kazemi	2020
2	Blanca	Lopez	2020
3	Cailey	Farrell	2020
4	Cesar	Bueno	2020
5	Chris	Amarvi	2020

ID -> First

ID -> Last

ID -> Start Year

Transitive functional dependency:

If you change the Instructor ID you need to change the Instructor Name. If you change Course Title you need to change both Instructor ID and Instructor Name.

Instructor Name is functionally dependent on Instructor ID. Instructor ID is functionally dependent on Course ID.

ID-> Instructor ID -> Instructor Name

Transitive dependency between course ID and Instructor Name

✔ **2nd NF**

Courses

ID	Course Title	Credits	Description	Instructor ID	Instructor Name
1	Java Introduction	4	Learn Java	1	Gregor
2	SQL Database Access	2	Write SQL	2	David
3	Spring MVC	2	Web development	3	Tara
3	REST Mirco Services	3	Write services	4	Josh

3rd NF – no other non key column needs to change based on the change of another non key column (ie: no transitive functional dependencies)

✔ **3rd NF**

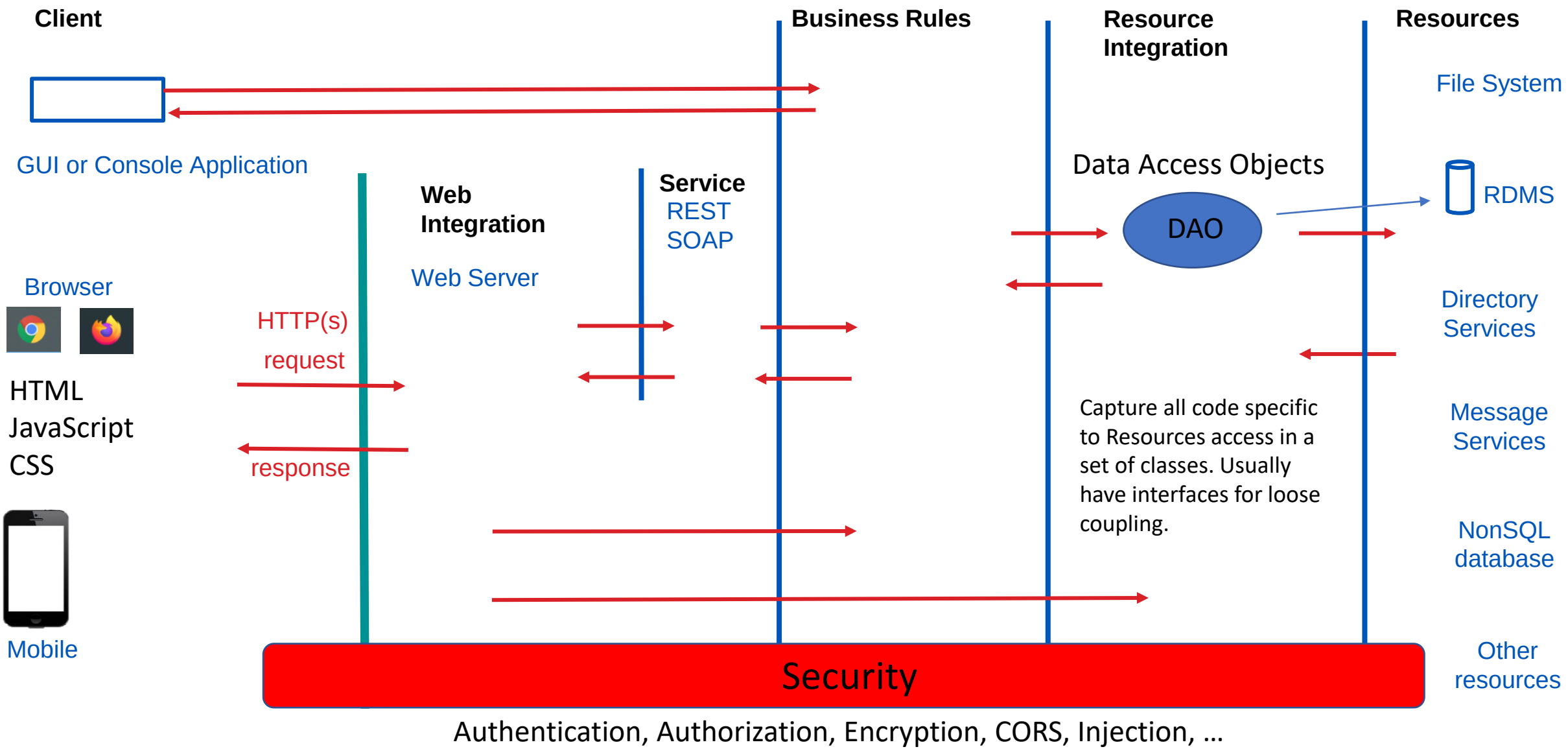
Courses

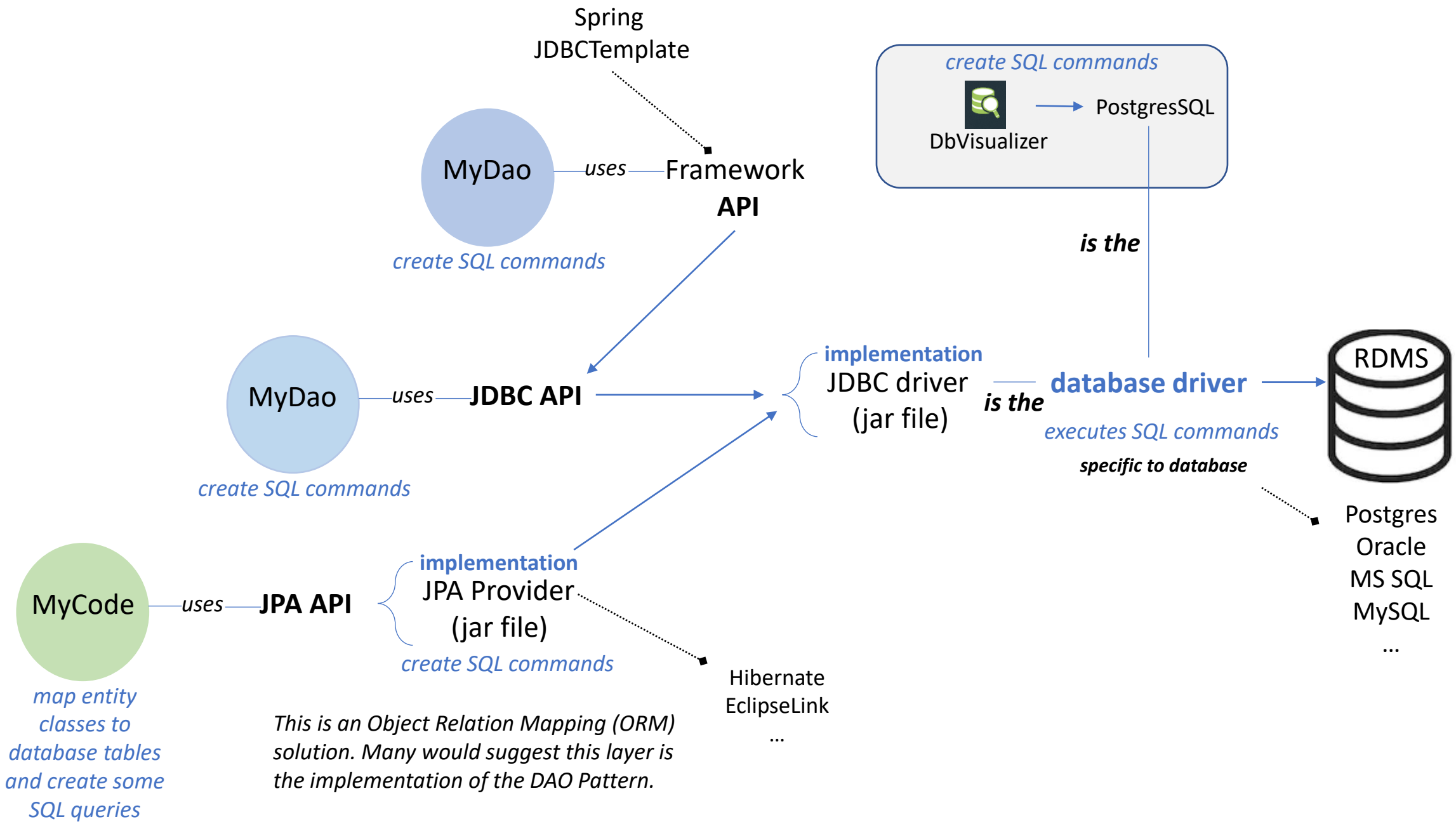
ID	Course Title	Credits	Description	Instructor ID
1	Java Introduction	4	Learn Java	1
2	SQL Database Access	2	Write SQL	2
3	Spring MVC	2	Web development	3
3	REST Mirco Services	3	Write services	4

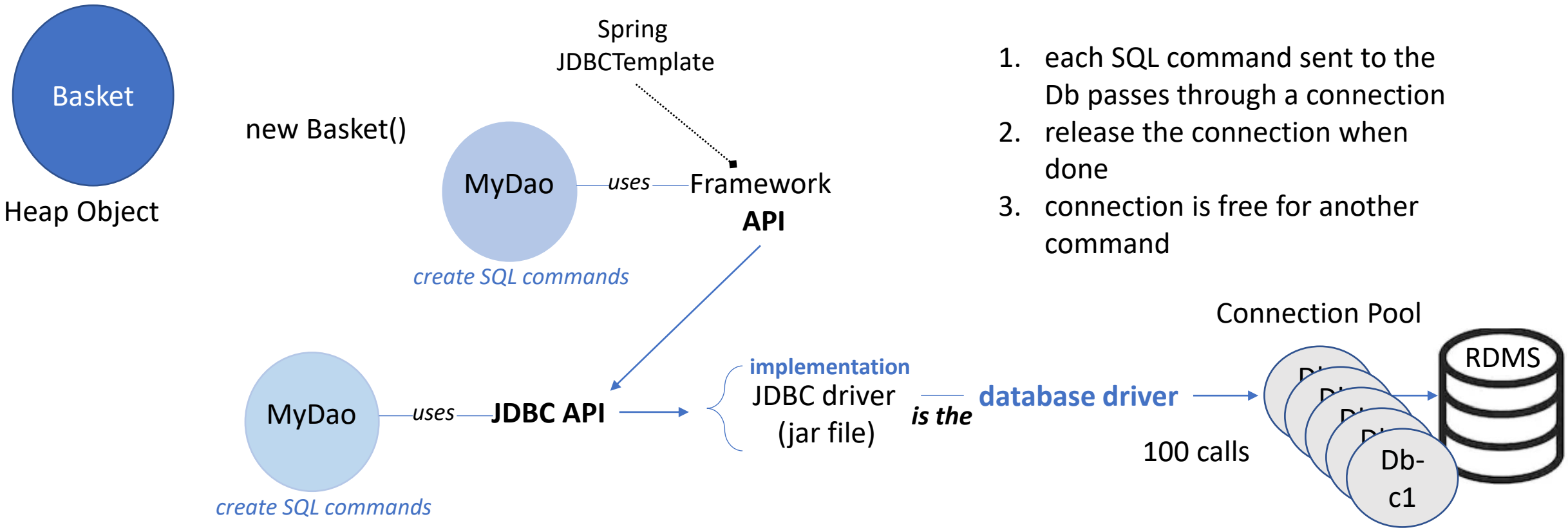
Instructors

Instructor ID	Instructor Name
1	Gregor
2	David
3	Tara
4	Josh

Application Architecture

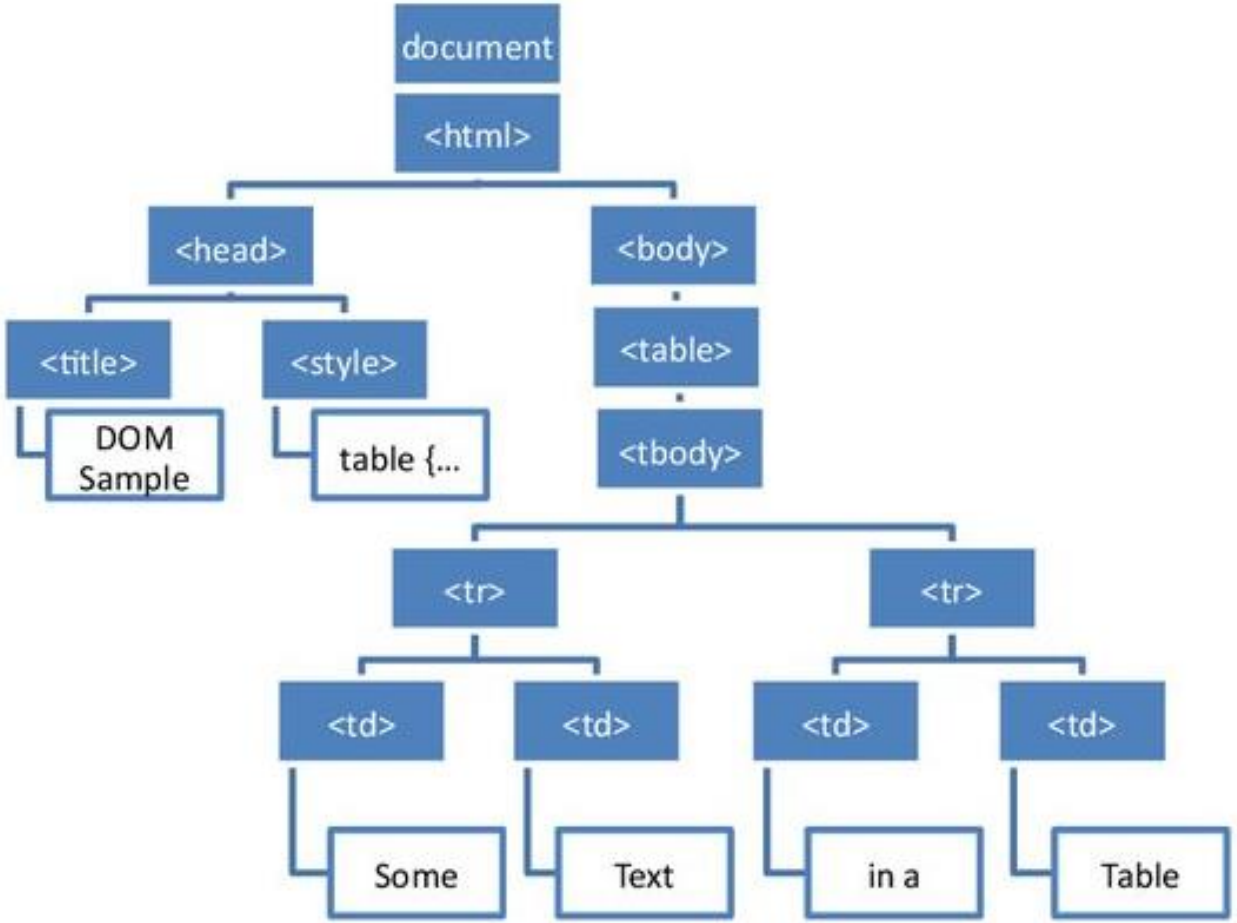



Spring
JDBCTemplate
MyDao
uses
Framework
API
create SQL commands
create SQL commands
DbVisualizer
PostgresSQL
is the
MyDao
uses
JDBC API
create SQL commands
implementation
JDBC driver
(jar file)
is the
database driver
executes SQL commands
specific to database
RDMS
Postgres
Oracle
MS SQL
MySQL
...
MyCode
uses
JPA API
implementation
JPA Provider
(jar file)
create SQL commands
Hibernate
EclipseLink
...
map entity
classes to
database tables
and create some
SQL queries
This is an Object Relation Mapping (ORM)
solution. Many would suggest this layer is
the implementation of the DAO Pattern.

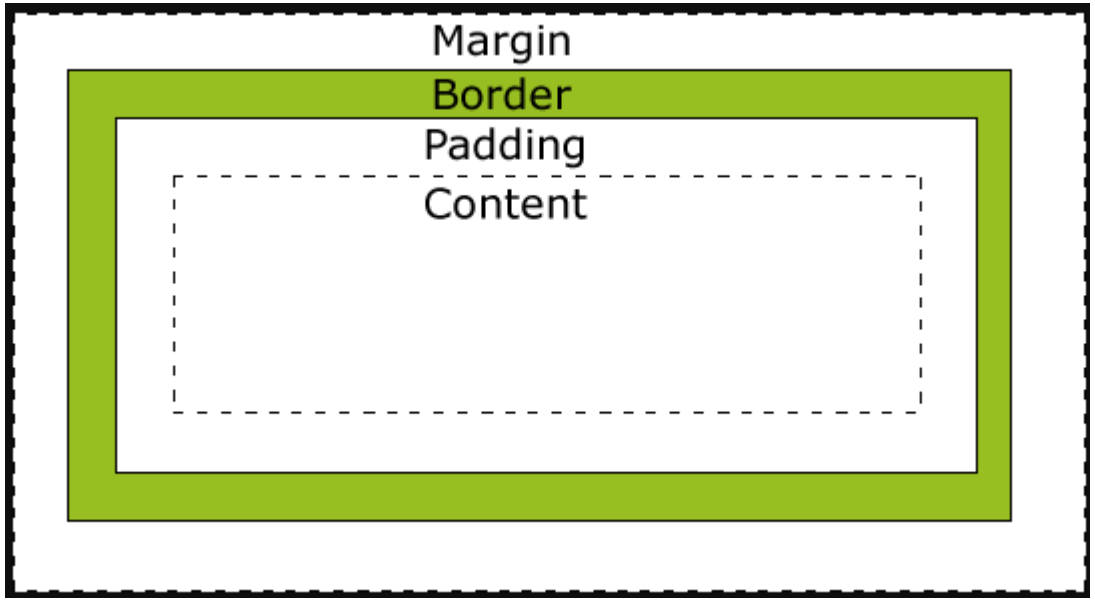


1. each SQL command sent to the Db passes through a connection
2. release the connection when done
3. connection is free for another command


```
<!DOCTYPE html>
<html>
    <head>
        <title>DOM Sample</title>
        <style type="text/css">
            table {
                border: 1px solid black;
            }
        </style>
    </head>
    <body>
        <table>
            <tbody>
                <tr>
                    <td>Some</td>
                    <td>Text</td>
                </tr>
                <tr>
                    <td>in a</td>
                    <td>Table</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```



Every element in web design is a rectangular box!



We use the content, padding, border, and margin to calculate the amount of space that an element takes up.

- Margin is the space **outside something**.
  - It does not affect the size of the box but affects other content that interacts with the box.

- Padding is the space **inside something**.

- The default width of a box isn't really 100% but less tangible *"whatever is left"*.
  - There are various circumstances where it is useful to **set** or **not set** a width.

Important HTML & CSS Related Definitions

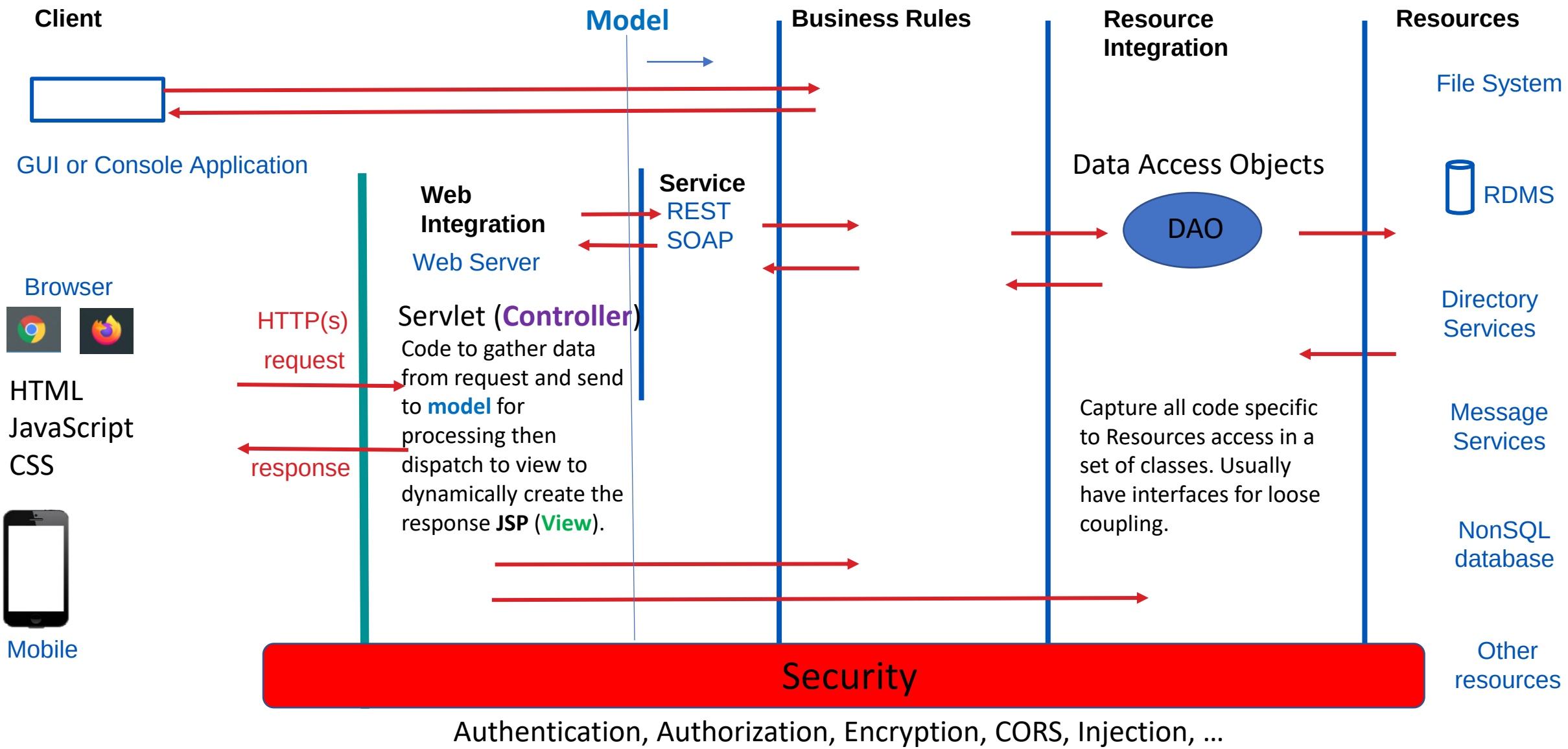
- Every HTML element has a default display value. For most HTML elements the default display value is either **block** or **inline**.
  - A **block-level element** always starts on a **new line** and takes up the **full width** available - * , <h1>-<h6>, <p>,* …
  - An **inline element** does not start on a new line and only takes up **as much width as necessary** - *, <a>, <img>,* …

- Can use *display:none* to hide an element. This is the default for the *<script>* tag.

- **Positioning**
  - Normal flow of a page is for elements to appear left to right, top to bottom based on the order in which they appear in the HTML document and the rules of block and inline display
  - **Static** (position by default) means the element will **conform to normal flow**
  - **Relative** position means **relative to where it would otherwise be positioned in the normal flow**
    - setting the *top, right, bottom, and left*
  - **Absolute** position will place the element relative to the parent ancestor (i.e. containing element) **exactly where you specify**
    - These elements are removed from the flow of the page.
    - Setting both *top* and *bottom*, or both *left* and *right*, you can stretch an element's dimensions.
  - **Fixed** position is **relative to the browser window** and does not scroll with the page.
    - setting the *top, right, bottom, and left*

- **CSS Variables -** enable programmers to define variables in CSS that can hold CSS Property Values

Application Architecture



URL identifies resource http://localhost:8080/myapp/css/site.css

protocol domain port application context path **resource**

webapp

- css
  - site.css
- images
  - logo.PNG
- js - files for JavaScript code
- profile - files and directories for profile features
- WEB-INF
  - classes - Java packages
  - jsp
    - common - jsp files common to pages like headers and footers
  - lib - jar files needed for the application
- index.html

A **resource** my be a request for a static file or a call to a method to create dynamic content.

jar utility → myapp.war → deploy

Tomcat

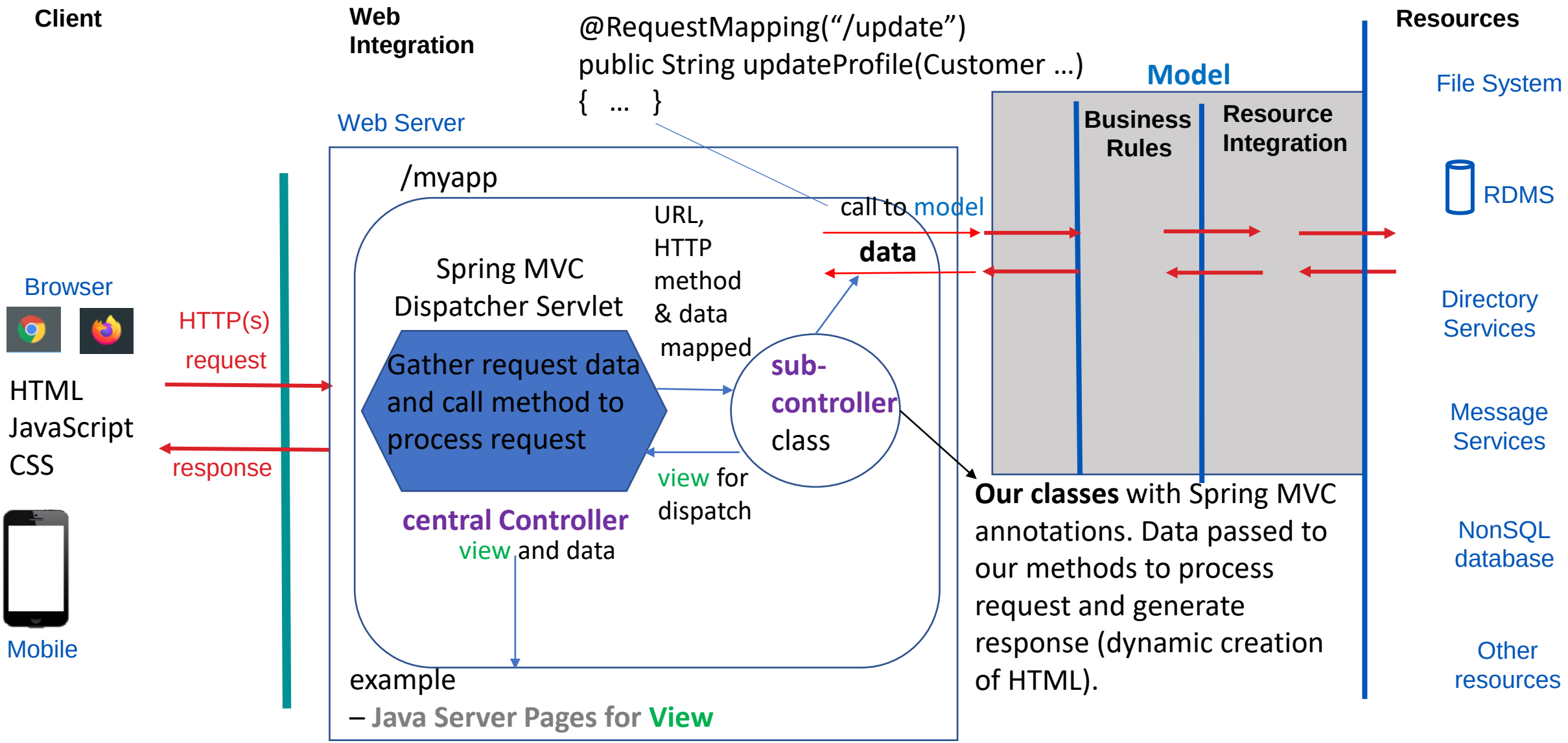
Application Server

- root.war
- myapp.war
- teller.war

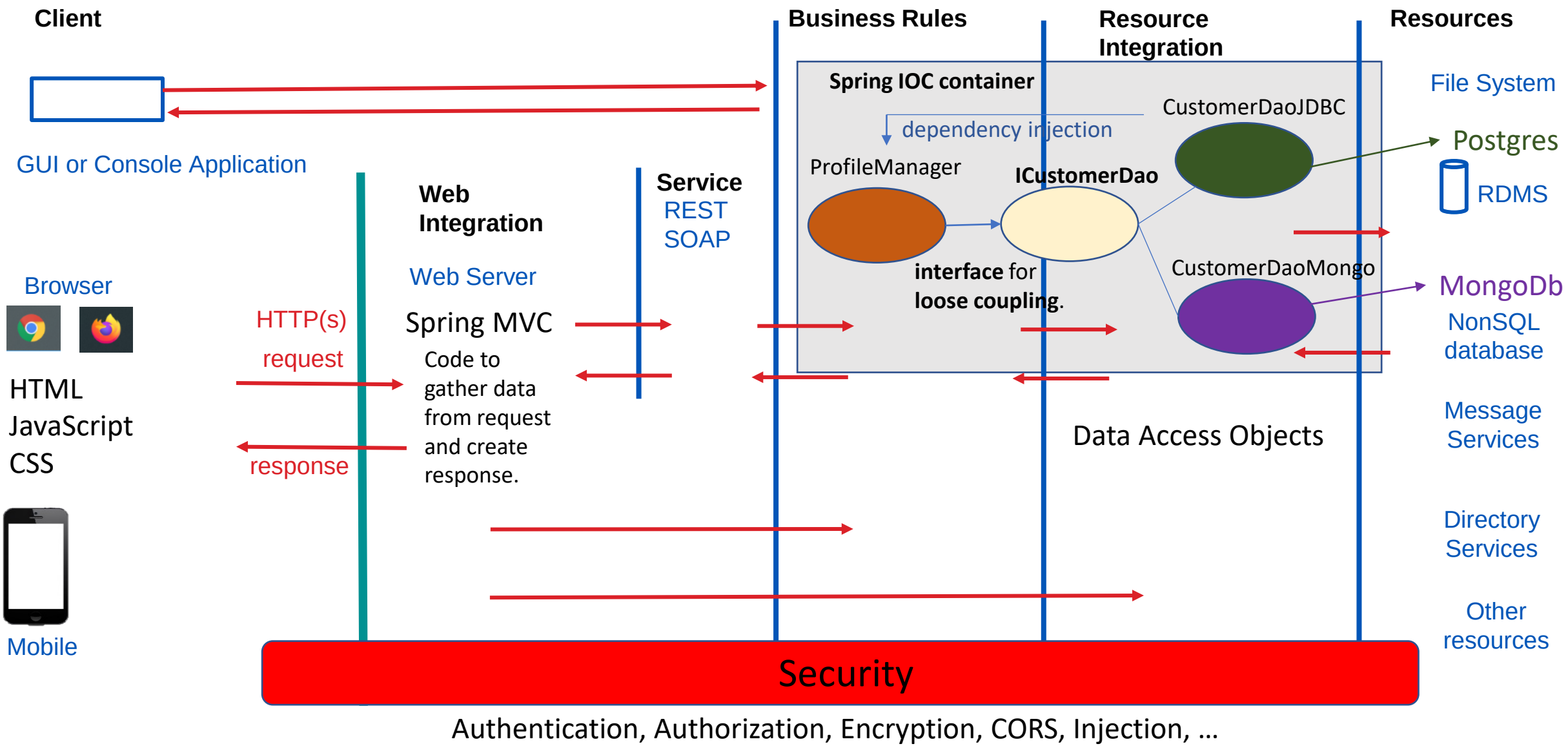
Application Architecture

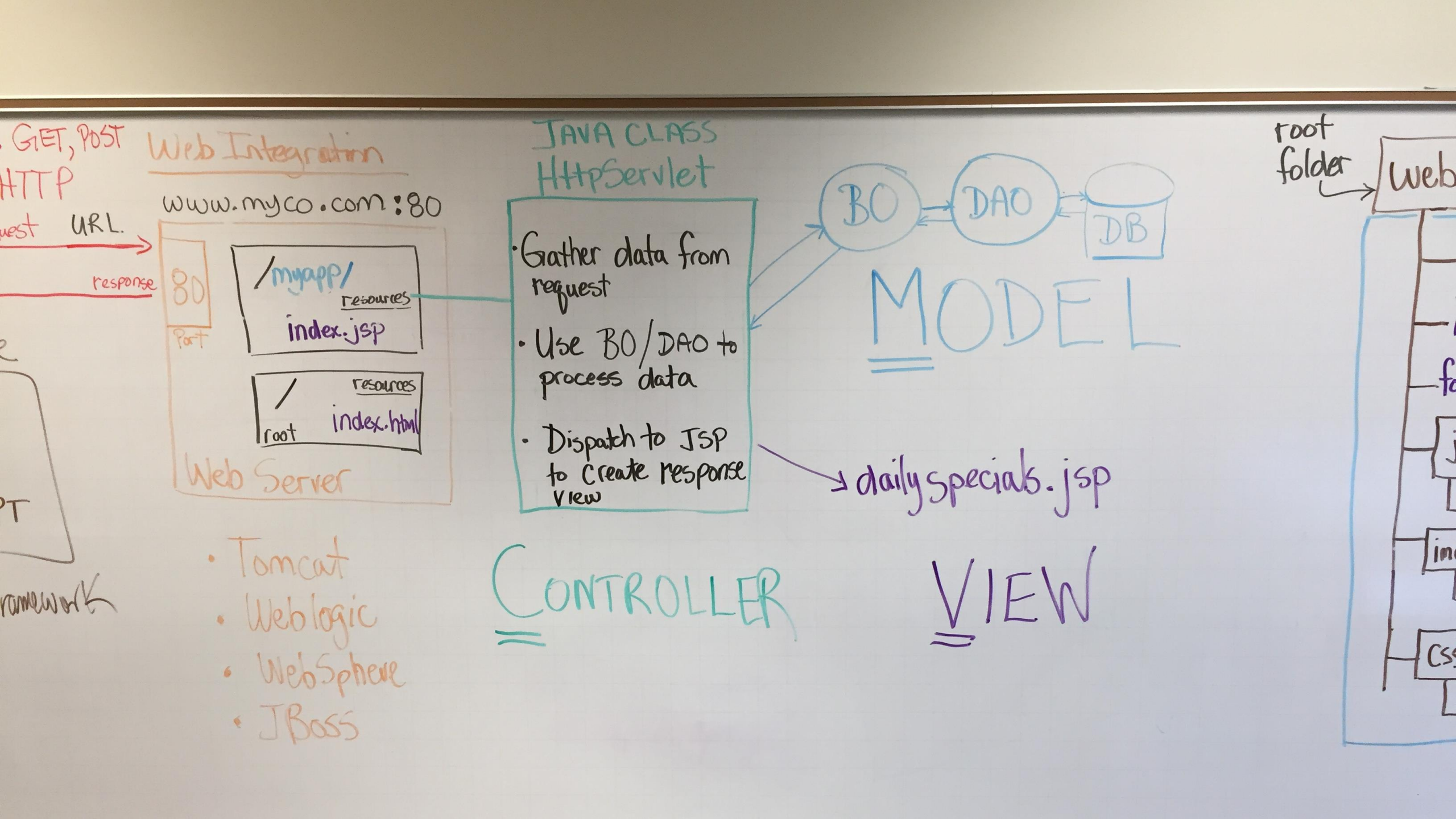
URL mapped sub-controller class method

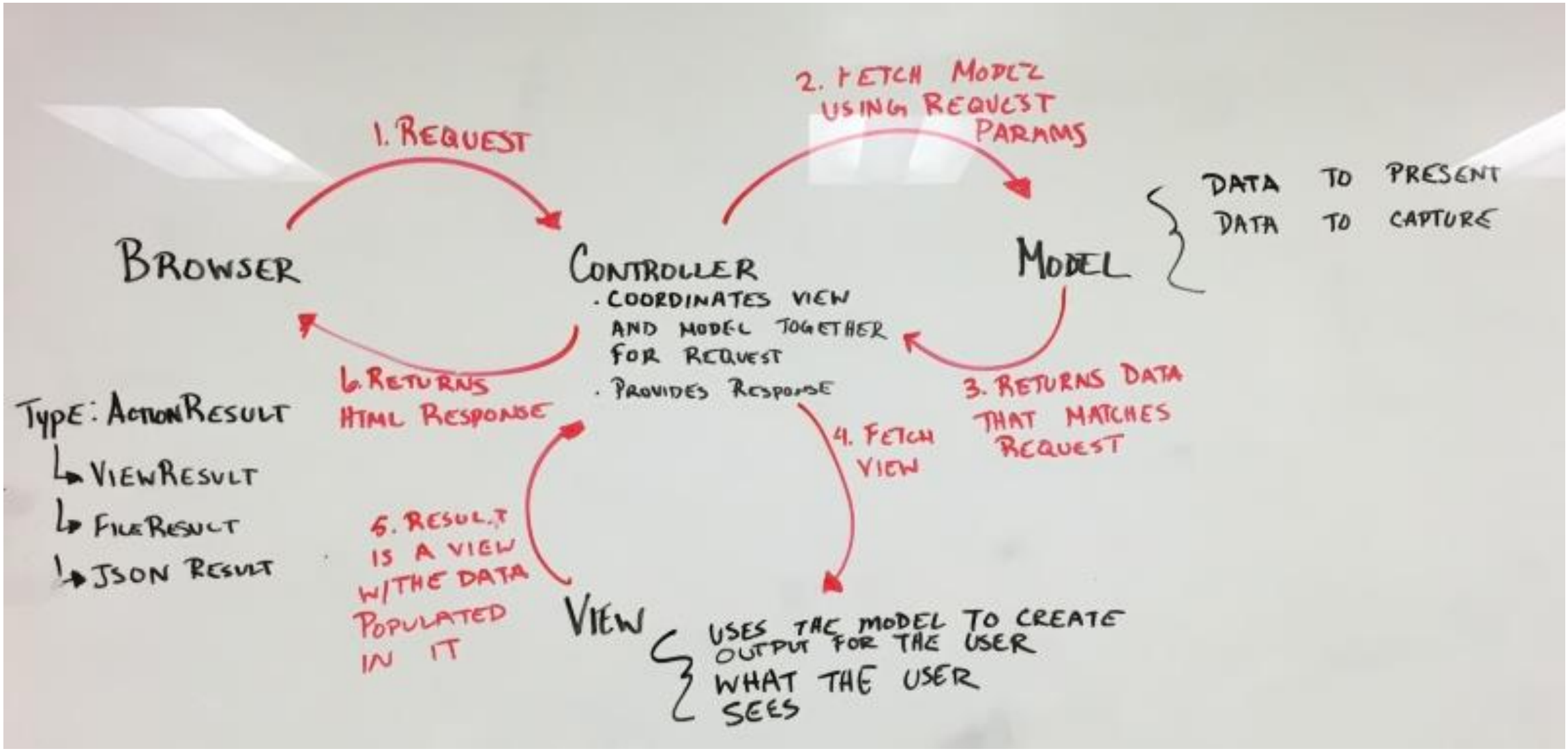
```
@Controller
class ProfileController
  @RequestMapping("/update")
  public String updateProfile(Customer ...)
  { ... }
```

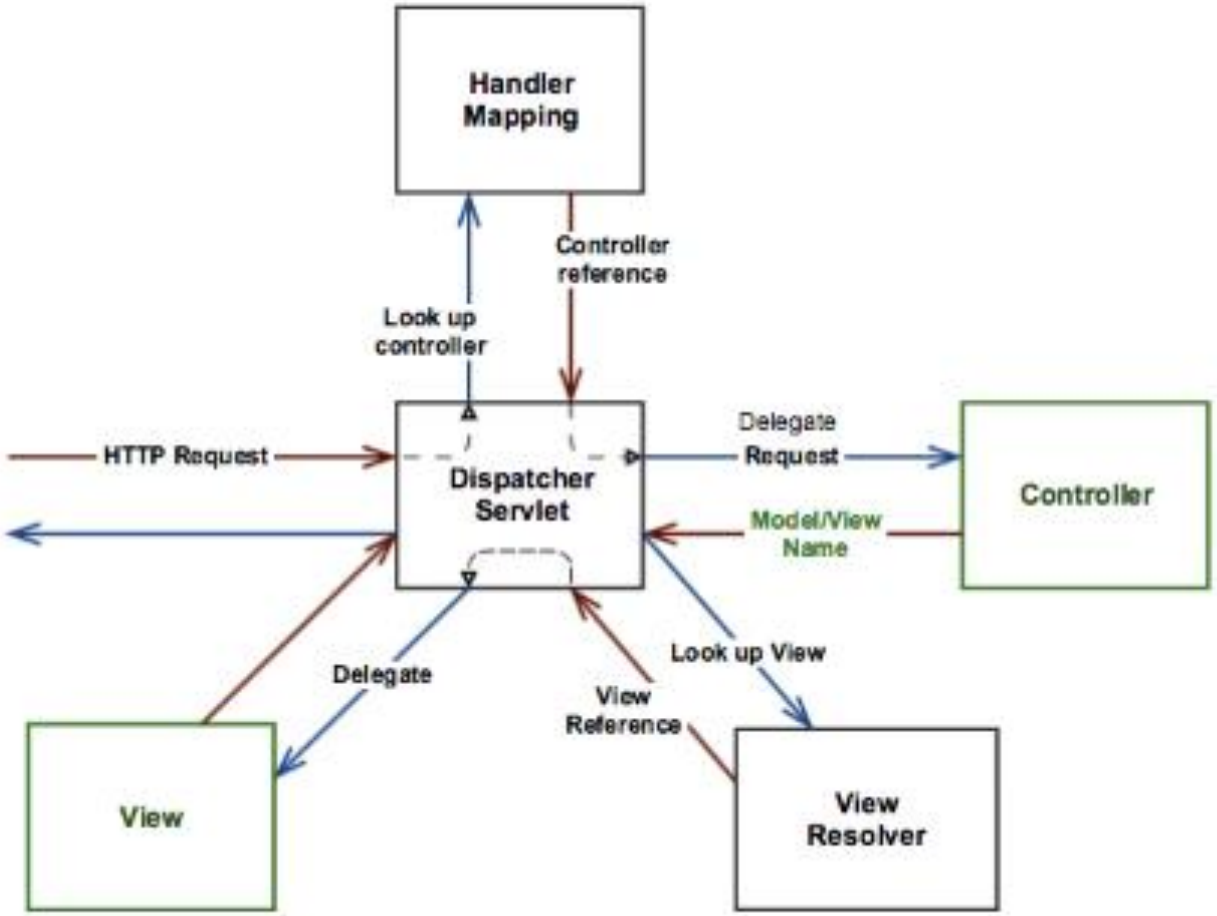


Application Architecture




GET, POST
HTTP
URL
response
Web Integration
www.myco.com:80
80
Port
/myapp/
resources
index.jsp
/
resources
root
index.html
Web Server
Tomcat
Weblogic
WebSphere
JBoss
JAVA CLASS
HttpServlet
Gather data from request
Use BO/DAO to process data
Dispatch to JSP to create response view
CONTROLLER
BO
DAO
DB
MODEL
dailyspecials.jsp
VIEW
root folder
Web


1. REQUEST
2. FETCH MODEL USING REQUEST PARAMS
BROWSER
CONTROLLER
· COORDINATES VIEW AND MODEL TOGETHER FOR REQUEST
· PROVIDES RESPONSE
MODEL
DATA TO PRESENT
DATA TO CAPTURE
6. RETURNS HTML RESPONSE
3. RETURNS DATA THAT MATCHES REQUEST
4. FETCH VIEW
5. RESULT IS A VIEW W/THE DATA POPULATED IN IT
VIEW
USES THE MODEL TO CREATE OUTPUT FOR THE USER
WHAT THE USER SEES
TYPE: ACTIONRESULT
VIEWRESULT
FILERESULT
JSON RESULT


Handler Mapping
Controller reference
Look up controller
HTTP Request
Dispatcher Servlet
Delegate Request
Controller
Model/View Name
Look up View
Delegate
View Reference
View
View Resolver

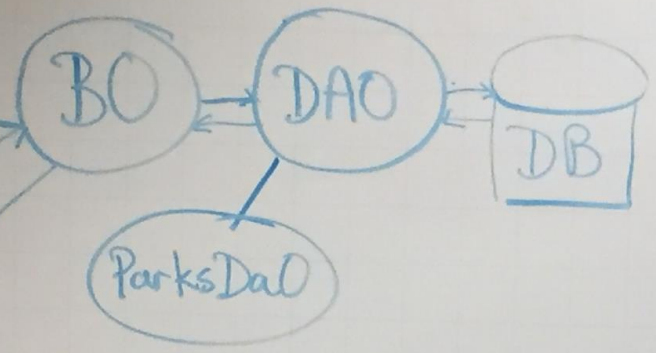
com:80

/
resources
.jsp

resources
index.html

HttpServlet

- Gather data from request
- Use BO/DAO to process data
- Dispatch to JSP to create response view



CONTROLLER
=
SPRING DISPATCHER SERVLET

/parks

/camps?id=1

@RequestParam

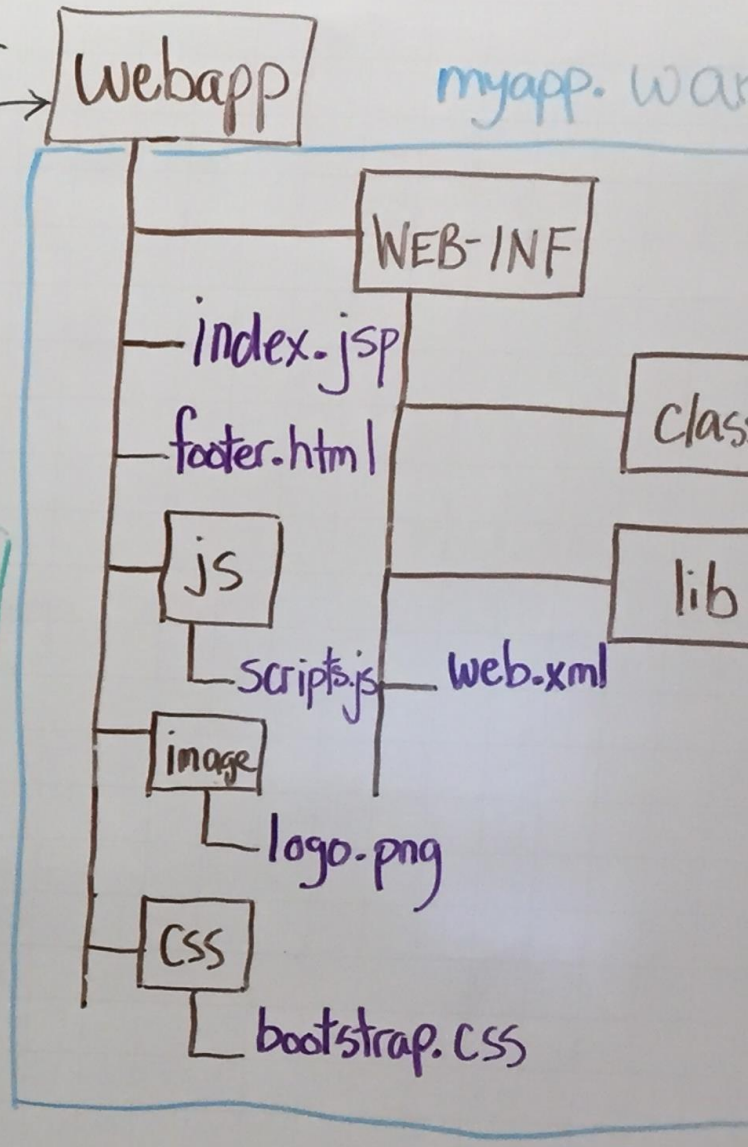
POJO

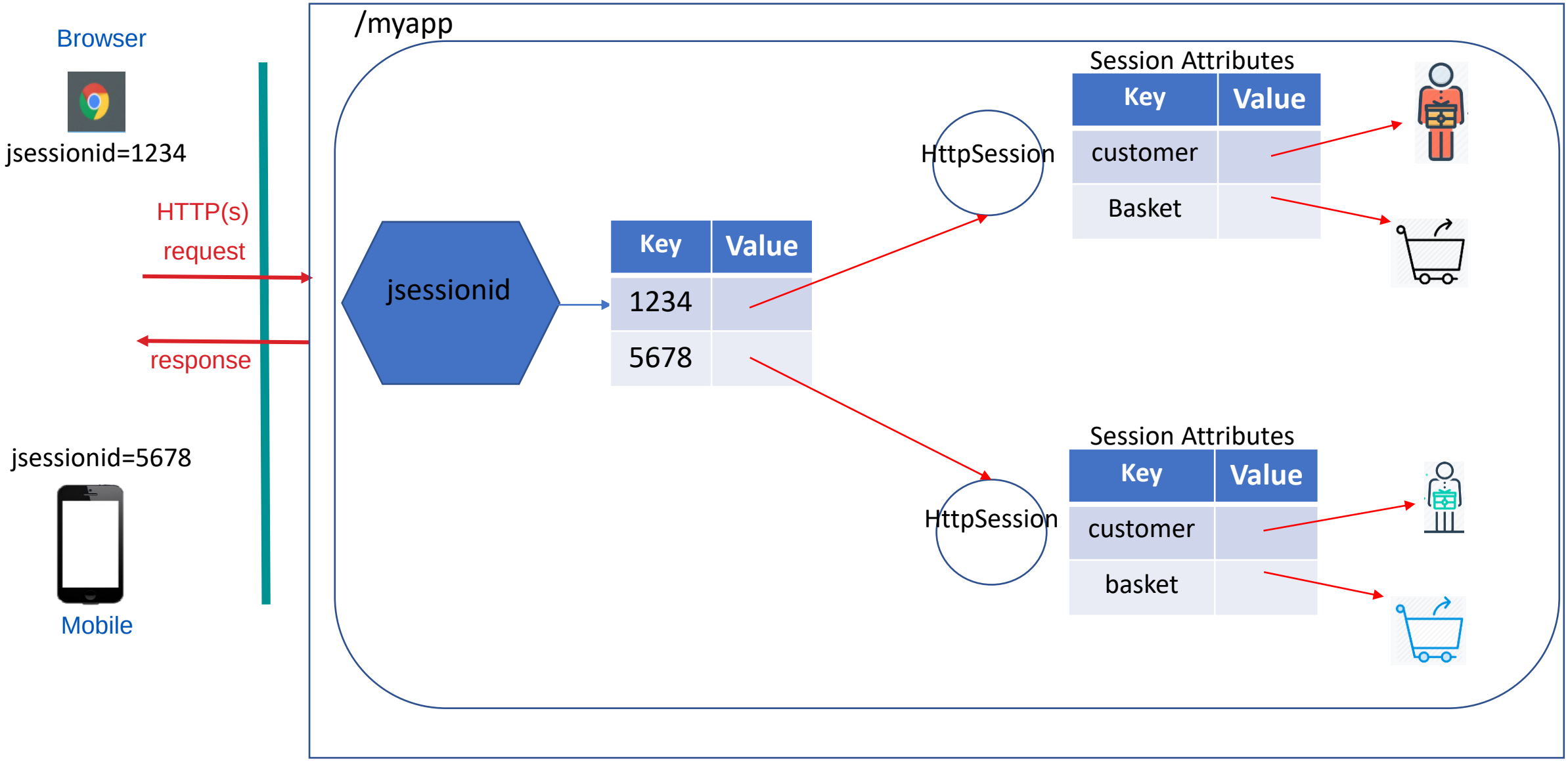
```
@Controller
public class ParkController

@RequestMapping("/parks")
String getParks()
    parksDao.getParks();
    return "parks";

@RequestMapping("/camps")
```

root folder




Browser
jsessionid=1234
HTTP(s)
request
response
jsessionid=5678
Mobile
/myapp
jsessionid
Key
Value
1234
5678
HttpSession
Session Attributes
Key
Value
customer
Basket
HttpSession
Session Attributes
Key
Value
customer
basket

Application Architecture