OOP - Day 5

Static vs non-Static

The Main Method

- Fixed method signature allows Java to start an application
- Notice it is marked `static`

```
public static void main(String[] args) {
  // ...
}
```

You've all seen the main method plenty of times by now. It is the place where a Java application begins. The reason Java knows where to begin, is because of the very strict signature of this method. The name, modifiers and arguments are fixed.

Other Methods

- Most methods do not have the `static` modifier
- Why is that?

```
public String getName() {
  // ...
}
```

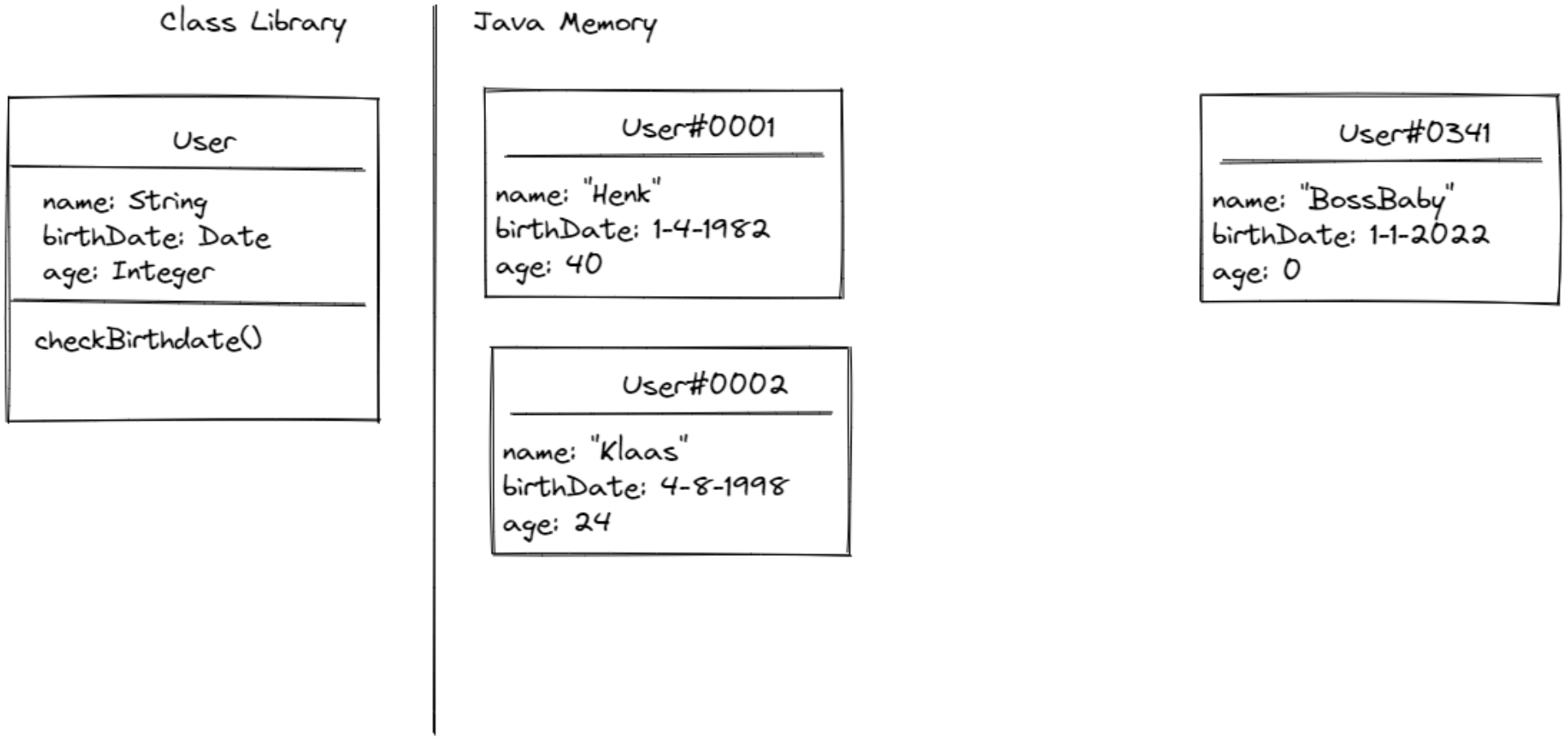
In Object Oriented programming, you want most of your methods to be non-static. In practice, this is also what we see in codebases. Why? With OO

we are keeping state and operations on that state together. In order for operations to be able to operate on state, they need to know *which* particular set of state variables they are working on.

What is an instance?

- When you write a class, you essentially create a template
- Java uses this template to reserve memory
- The reserved memory will hold all you class variables
- This bit of memory is called an *instance*

Instance in Memory



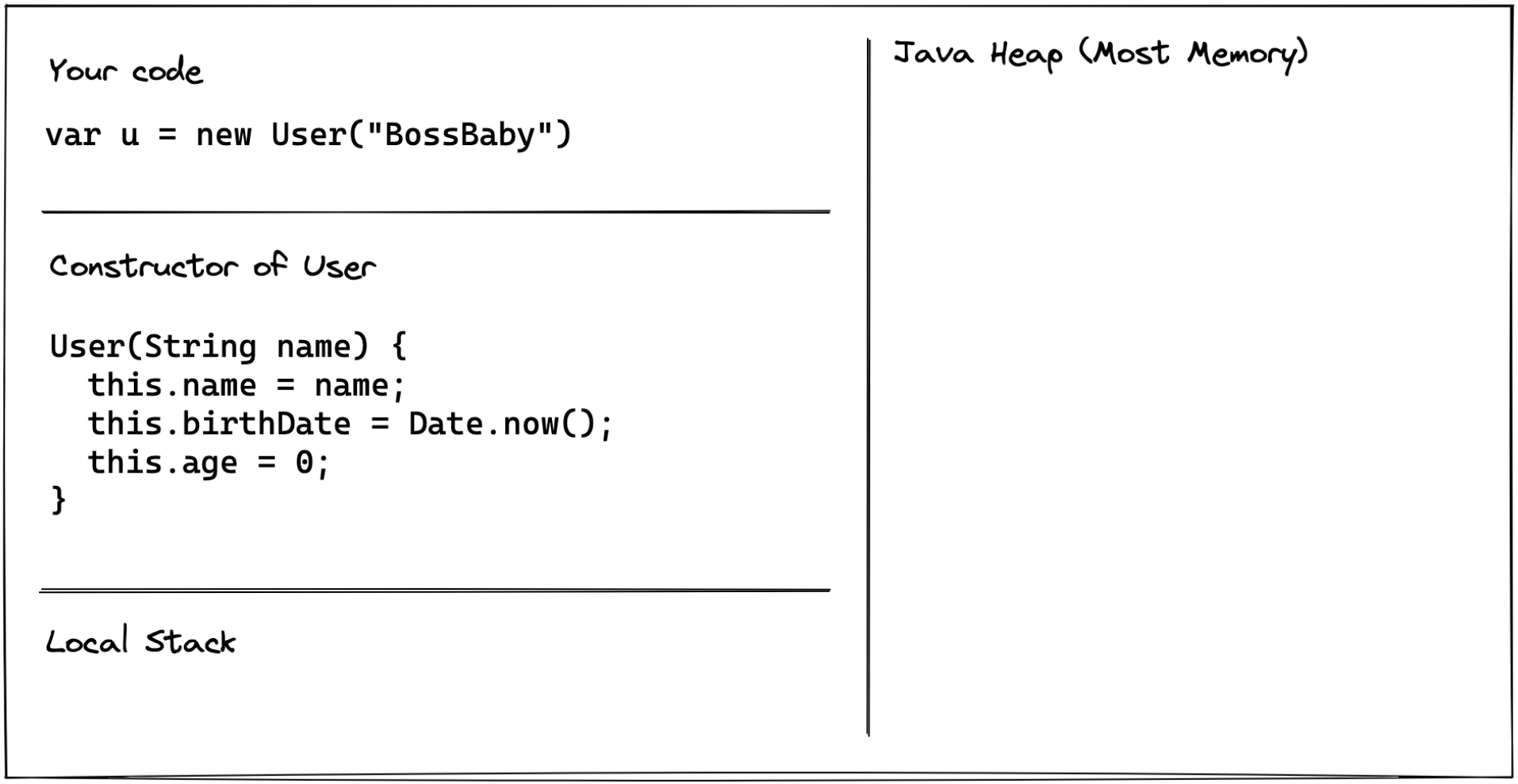
Creating an Instance

- A constructor creates a new *instance*
    1. Space for variables is reserved
    2. Constructor is called to initialize variables

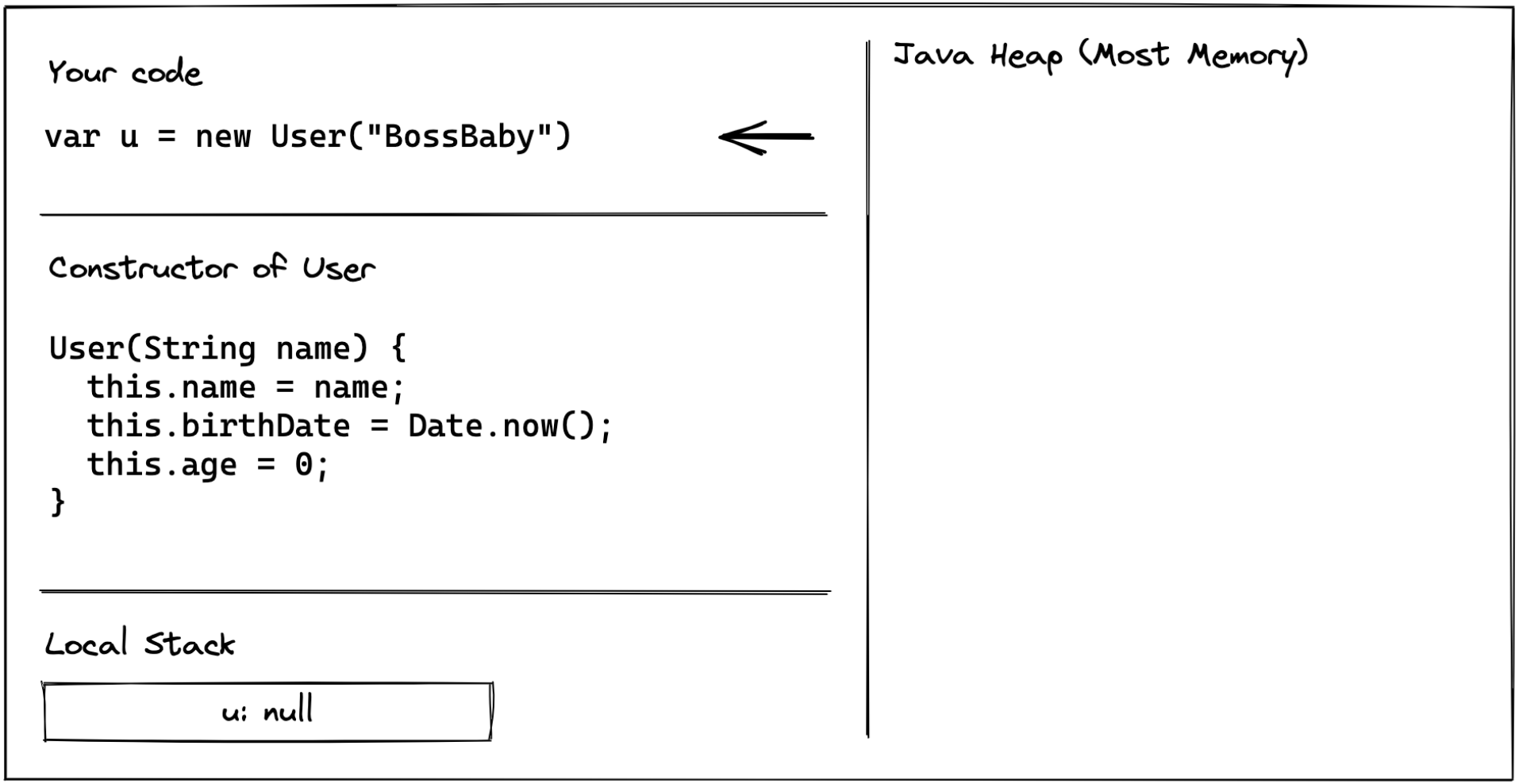
3. The reference is stored in your local variable

When a constructor is called, Java uses the class as a template to fill in a new piece of memory. This new piece of memory is often referred to as an `Object`.

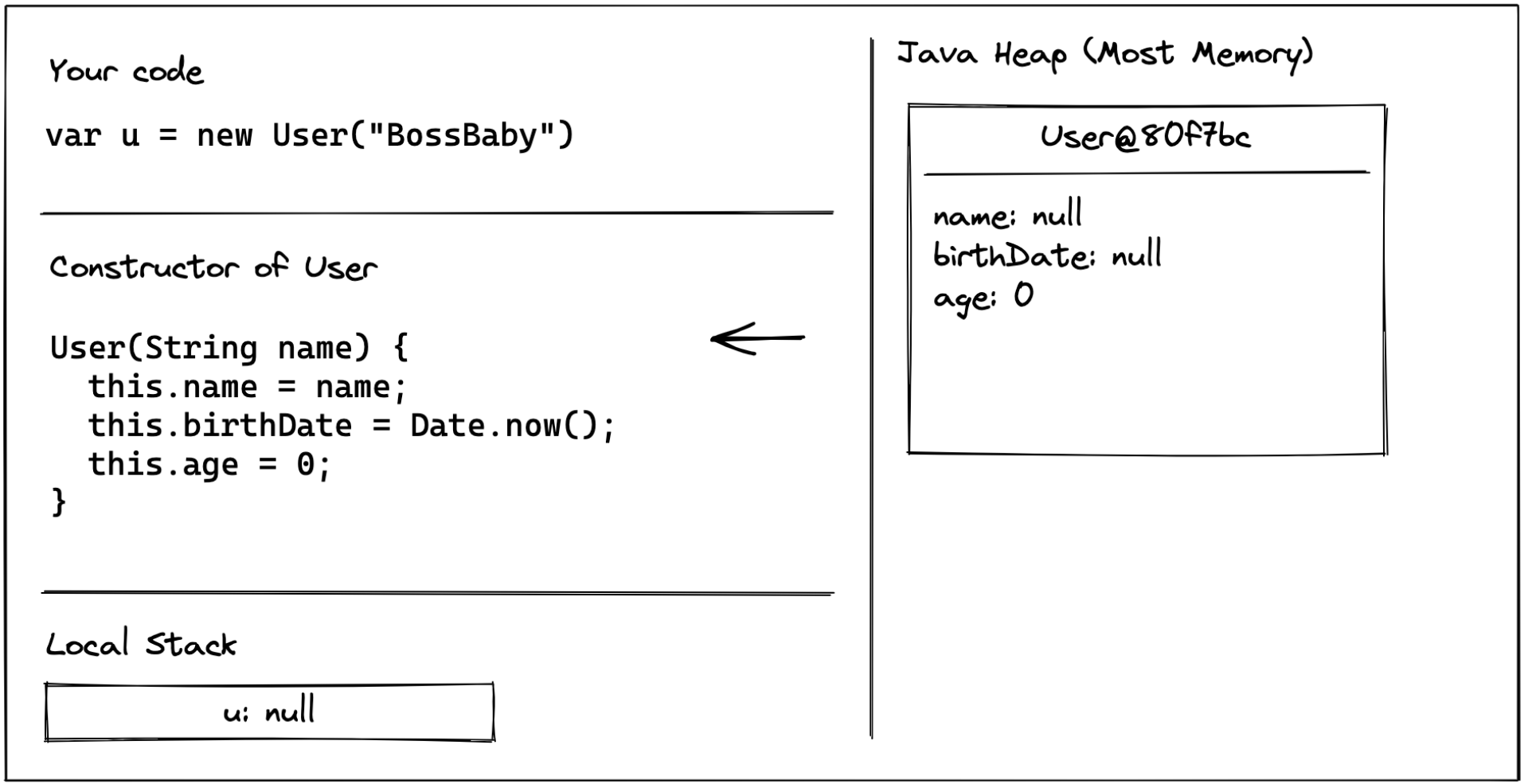
Creating an Instance



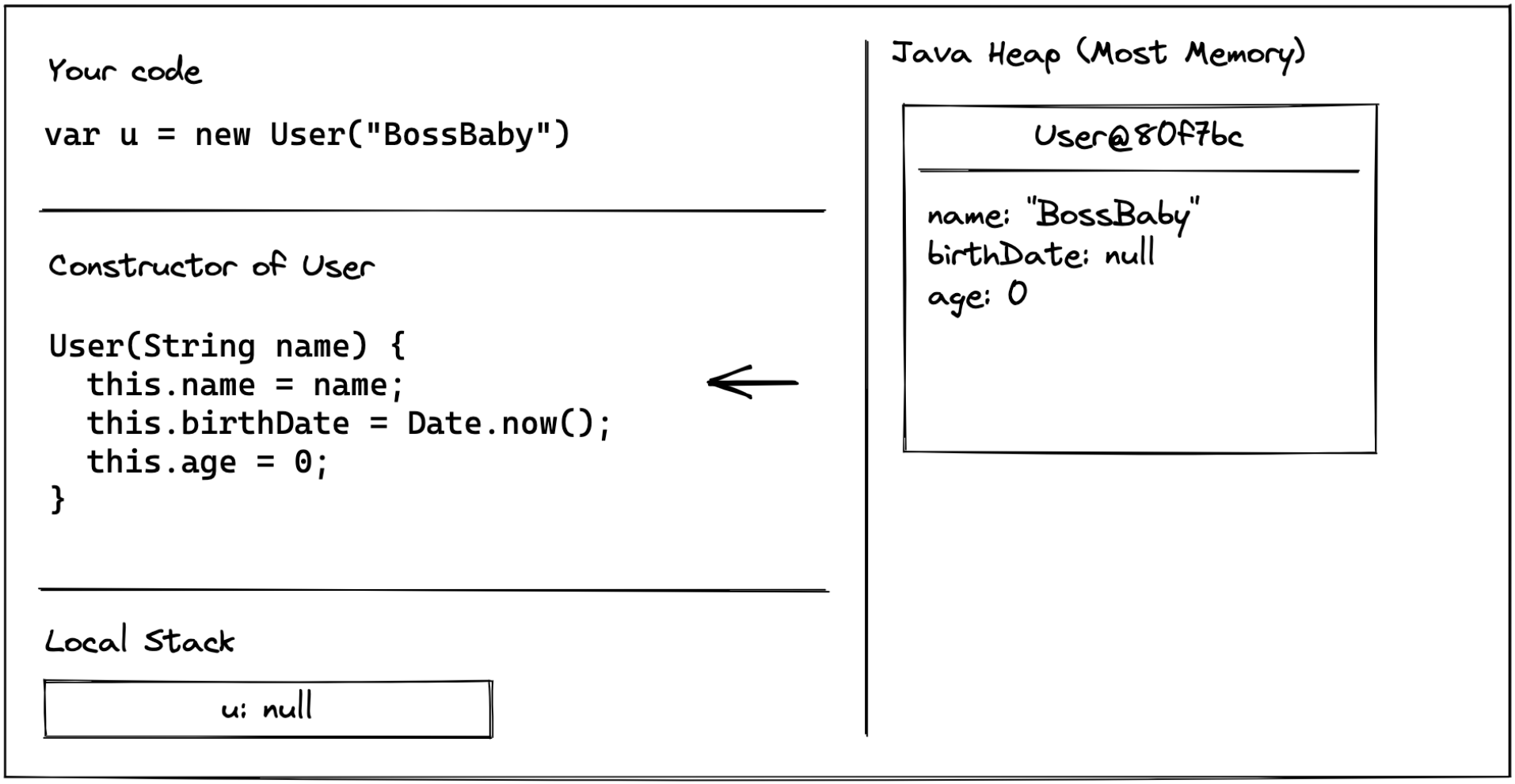
Creating an Instance



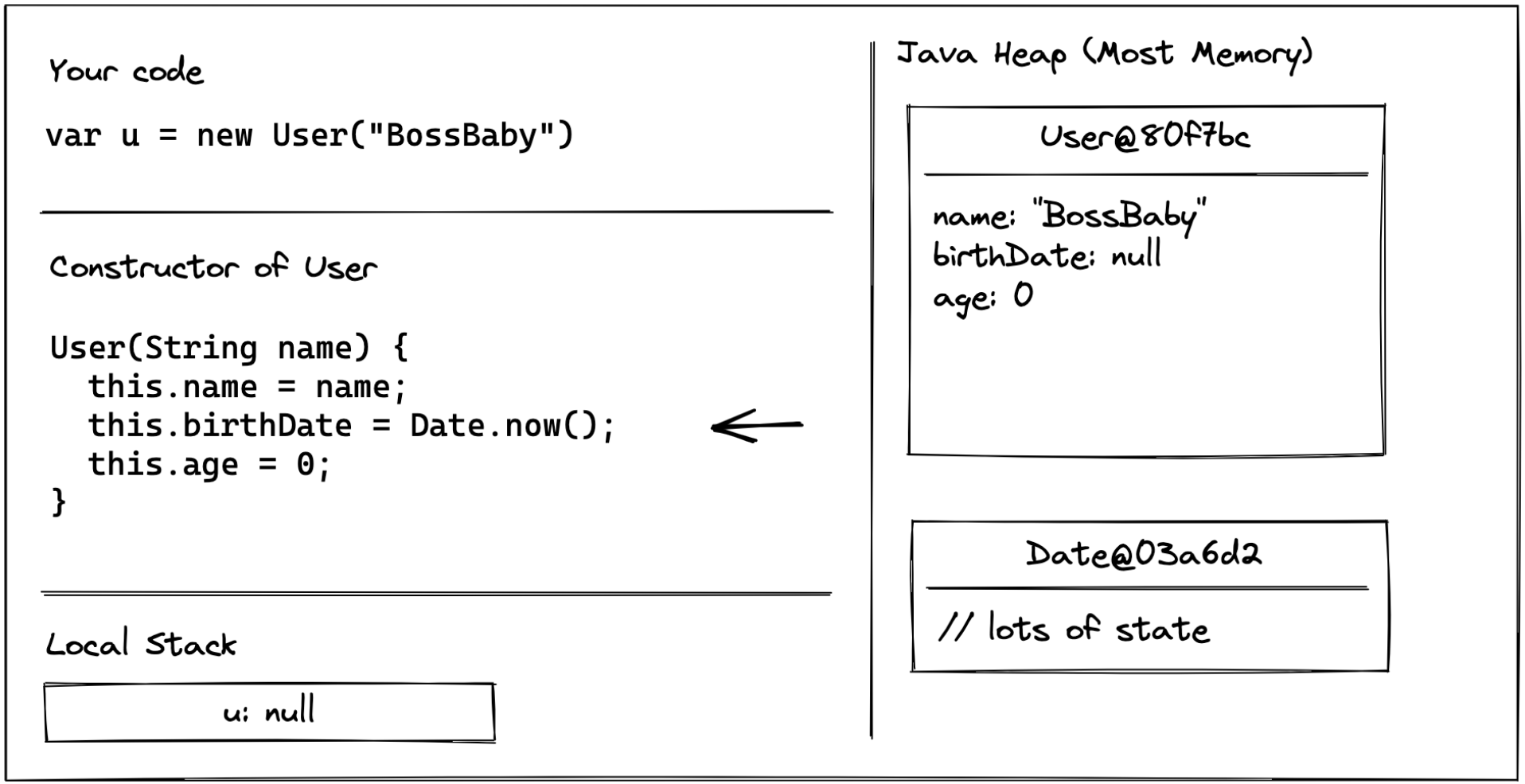
Creating an Instance



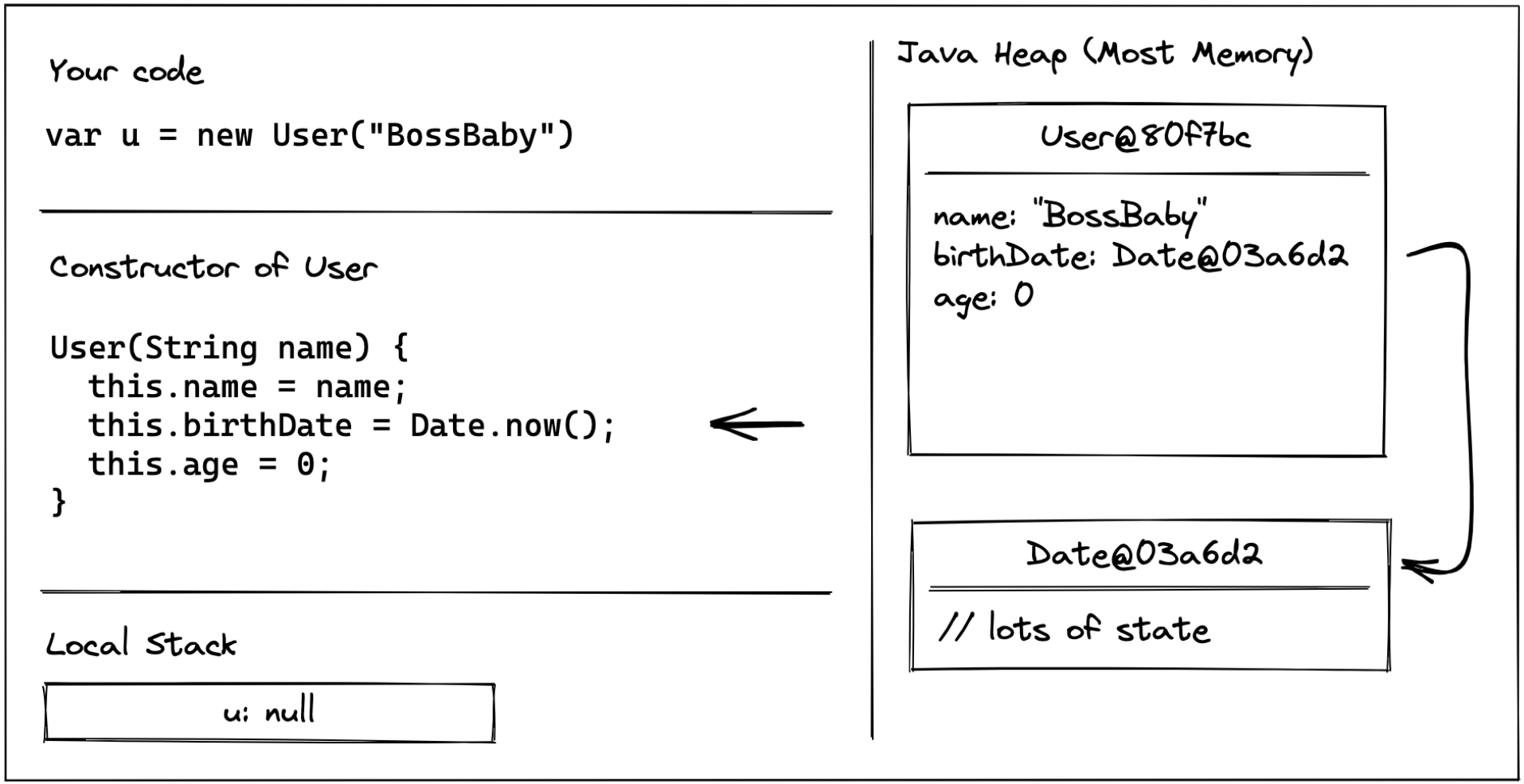
Creating an Instance



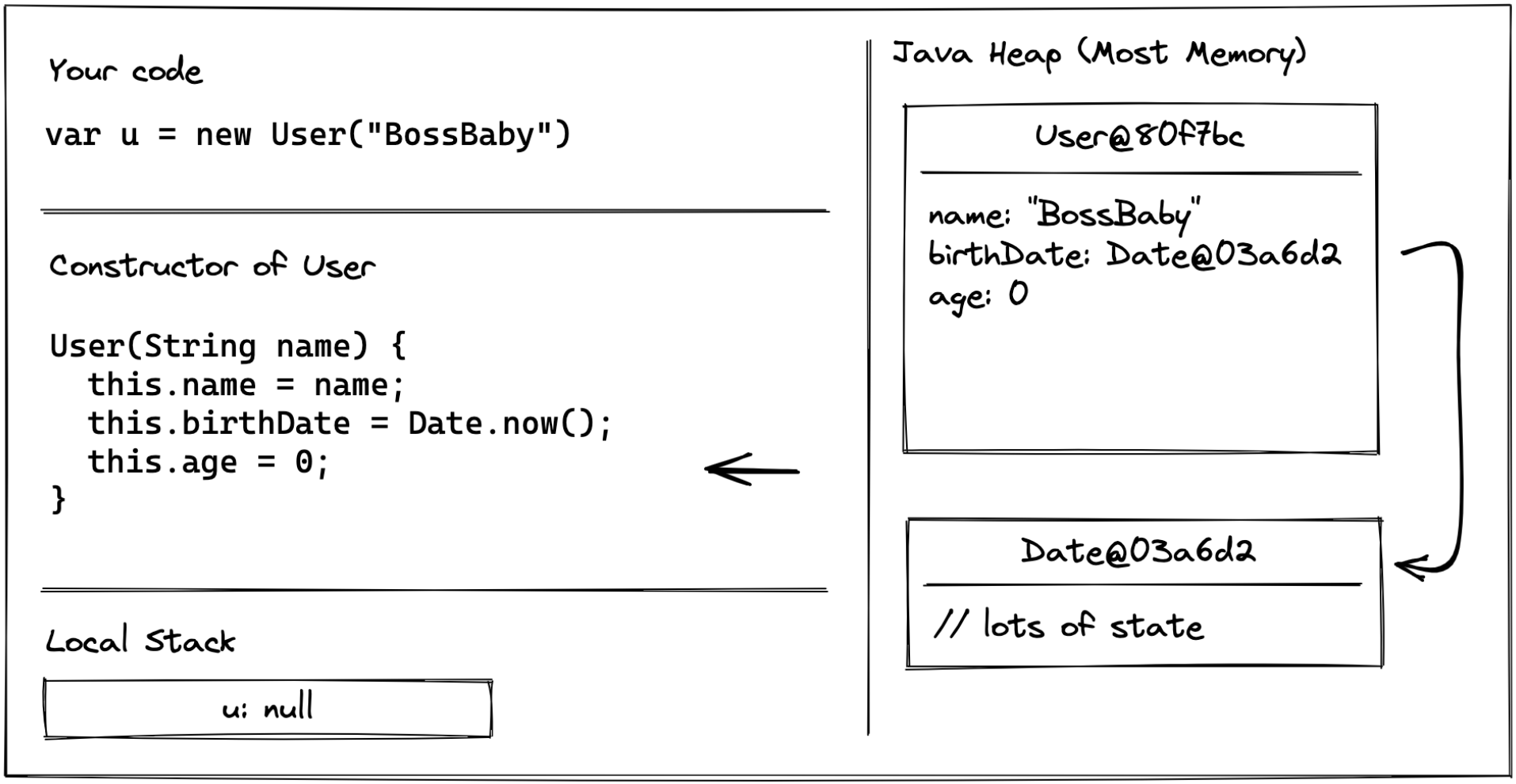
Creating an Instance



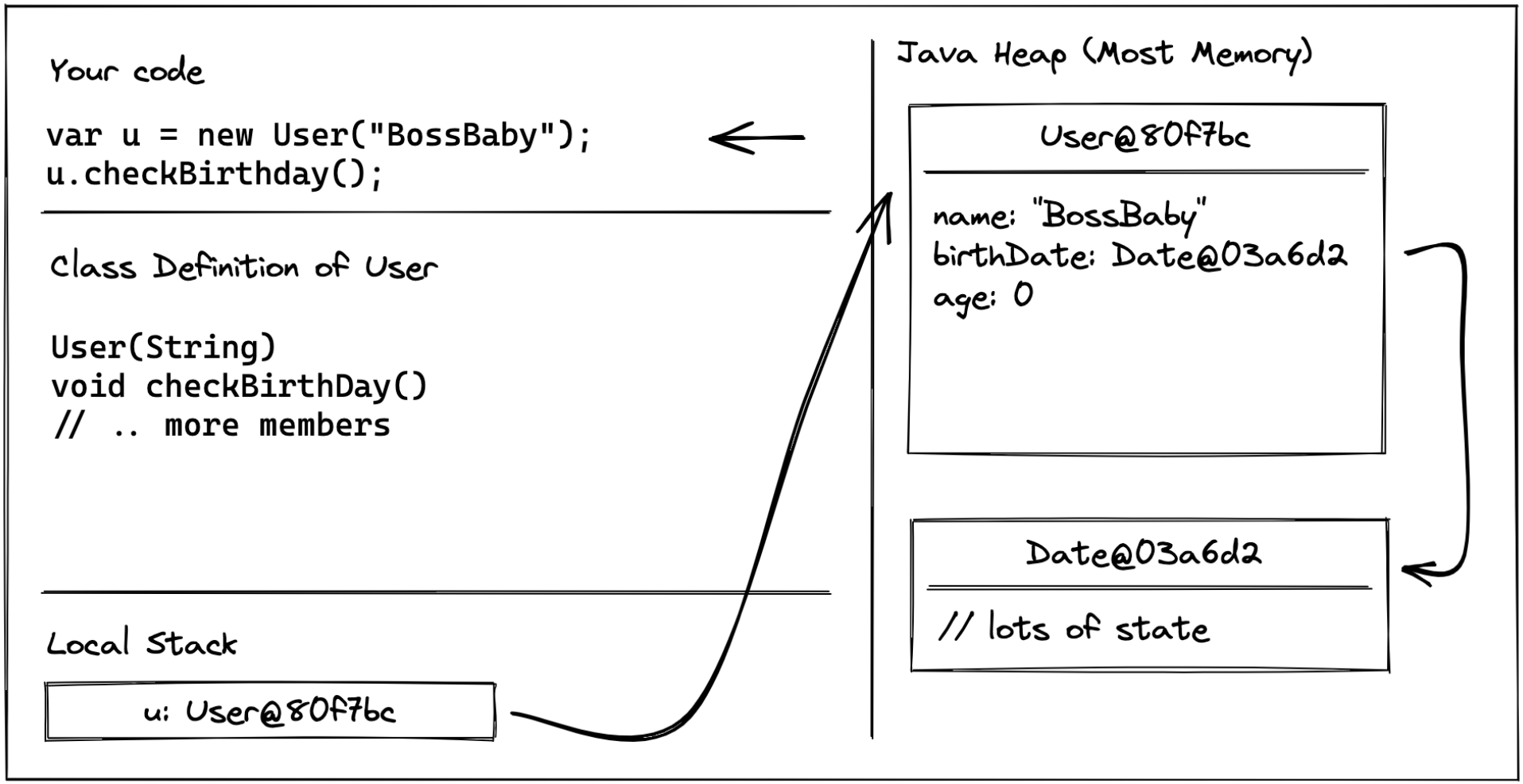
Creating an Instance



Creating an Instance



Creating an Instance



Instance Methods

- Need an *instance* to work with `this`

- Can use variables scoped or non-scoped:

```
// Equivalent
System.out.println(this.name);
System.out.println(name);
```

- Only needed when there more `name` variables

```
class User {
  String name;

  User(String name) {
    this.name = name;
  }
}
```

Static Methods

- Static methods have no `this` and no instance
- They cannot use `this`
- You can call them without an instance
- When your application starts, there are no instances
- This is why `main` is `static`

Introduction to OOP

Programming Paradigms

- Imperative Programming
    - Developer declares exact sequence of instructions to execute

```
int a = 0, b = 2;

a = b * 4;
b = a * a;

System.out.println(b);
```

Programming Paradigms

- Declarative Programming
    - Developer declares intent, instructions are derived by the platform

```
<p>This is a paragraph, a small piece of text that gets rendered
   as a whole unit. Styling depends on the browser and CSS files</p>
<input type="text" name="yourName">
<button name="sendYourName">
```

Programming Paradigms

- Functional Programming
    - Models functions closely to mathematics

```
def square = (x: Int) => x * x
def moduloTen = (x: Int) => x % 10

def squareModuloTen(x: Int): Int = moduloTen(square(x))
// This is the same, but uses function composition
def squareModuleTen(x: Int): Int = (square andThen moduloTen)(x)
```

Programming Paradigms

- Object Oriented Programming
    - Identifies things in the real world as objects

```
class Person {
  String name;
  int age;

  void celebrateBirthday() {
    System.out.println("Happy birthday!");
    this.age += 1;
  }
}
```

Other Paradigms

- Procedural Programming
- Event Driven Applications
- Processing Pipelines

Java’s Paradigms

- Object Oriented
- Imperative
- (Some Functional Features since Java 8)

Funny OOP Abbreviations

OOP

Object Oriented Programming

OOPS

Object Oriented Programming Systems

OOPLA

Object Oriented Programming Language(s)

JOOP

Java Object Oriented Programming

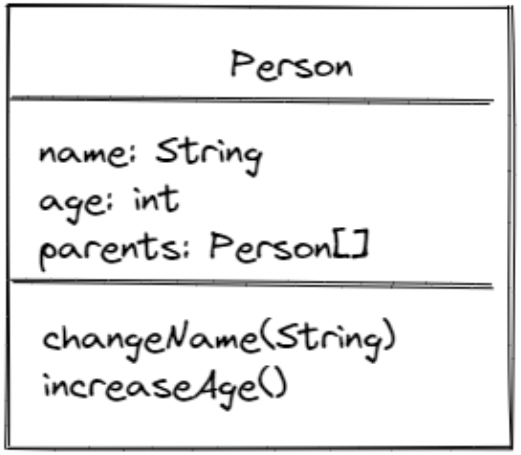
Concepts of OOP

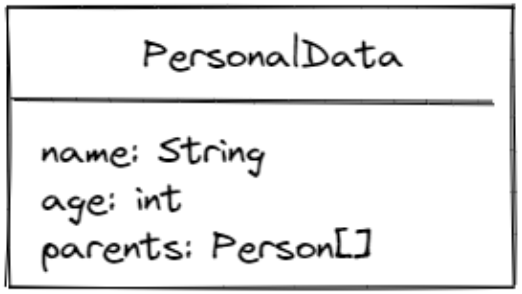
- Classes
- Objects / Instances

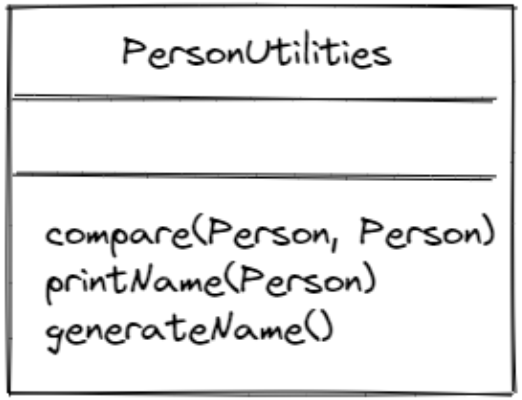
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Message Passing / Method Calling

Concept: Classes

- Central OOP Concept
- Groups data and behaviour
- Typical class has fields and methods

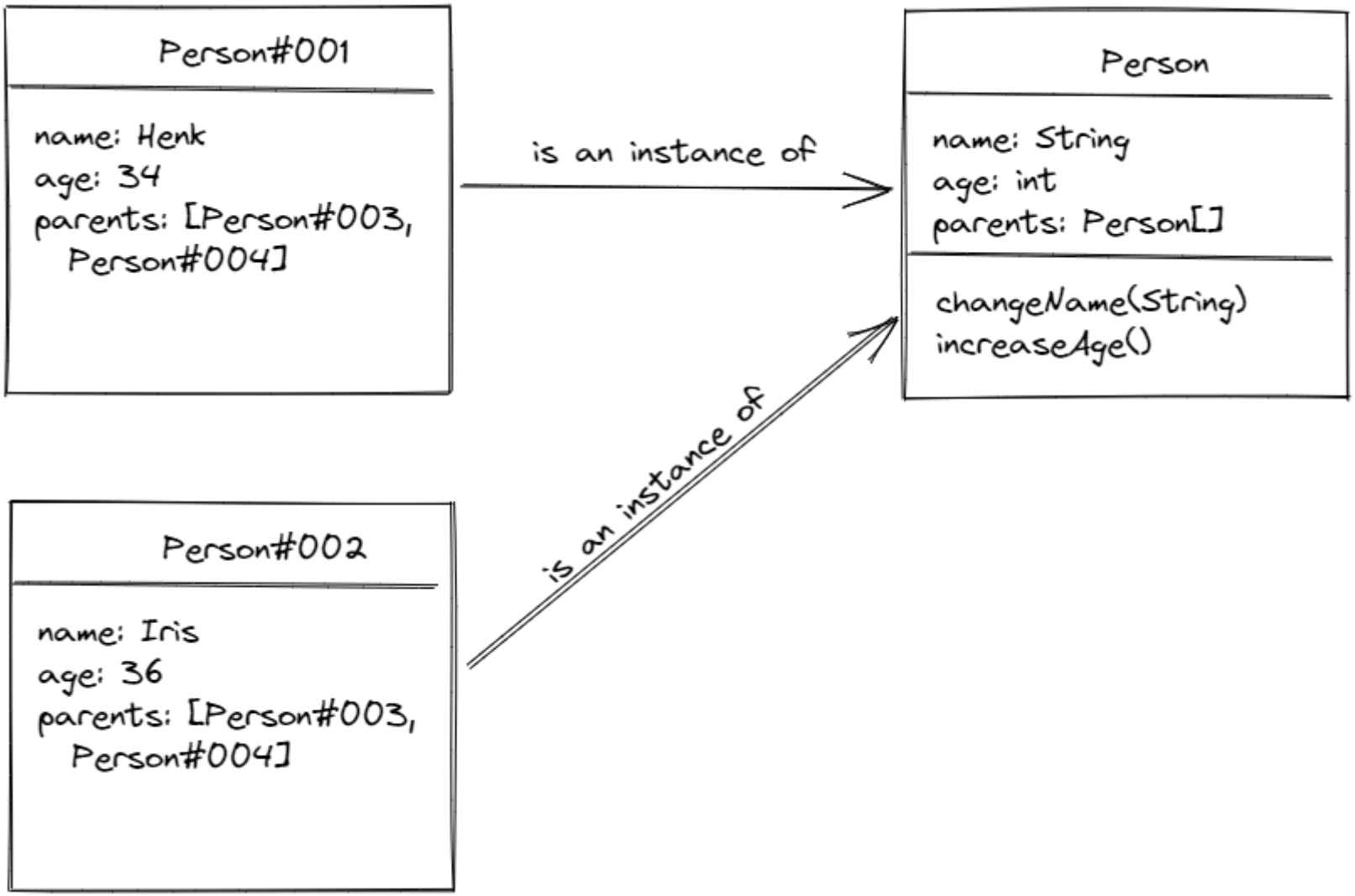






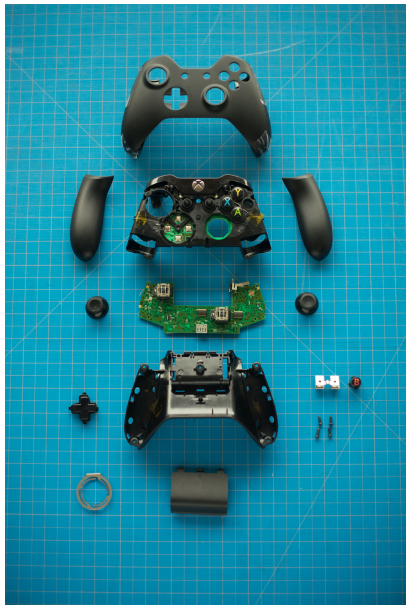
Concept: Objects / Instances

- A class describes structure
- An Object / Instance …
    - occupies space in memory
    - holds the data
    - points to its class
    - can have multiple Instances per class


Person#001
name: Henk
age: 34
parents: [Person#003,
Person#004]
is an instance of
Person
name: String
age: int
parents: Person[]
changeName(String)
increaseAge()
Person#002
name: Iris
age: 36
parents: [Person#003,
Person#004]
is an instance of

Concept: Data Abstraction

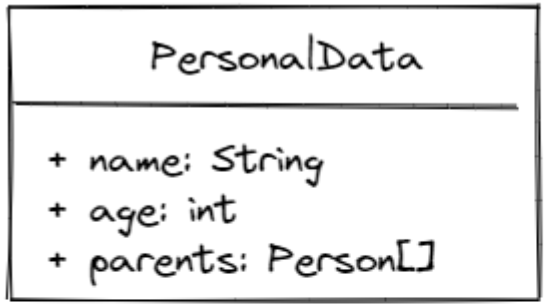
- Method to reduce complexity
- Select properties and behaviours important to the solution
- Leave other details out



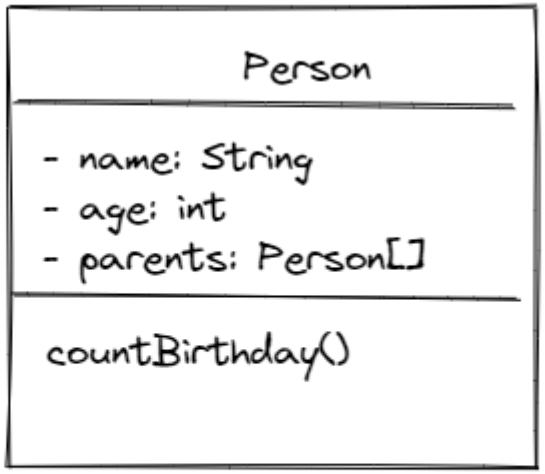
Concept: Encapsulation

- Groups data into single unit
- Strictness levels:
    - Low: data is grouped, but not protected

- High: data is protected, altered through methods



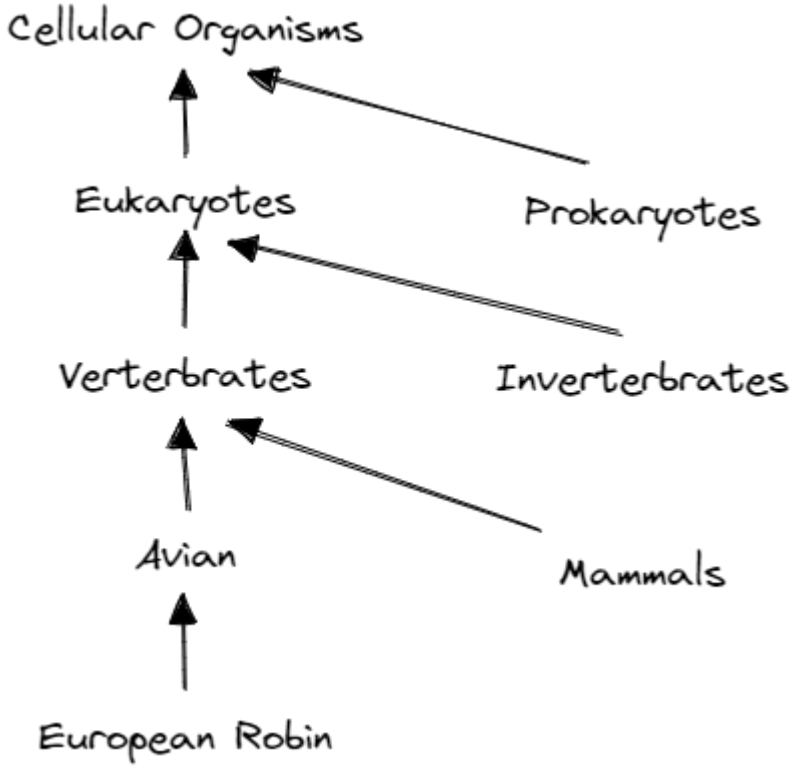

```
// Updating age
person1.age += 1;
```




```
// Updating age
person1.countBirthday();
```

Concept: Inheritance

- Helps minimize duplication
- Fields and methods are inherited from parent(s)
- Hierarchies of classes



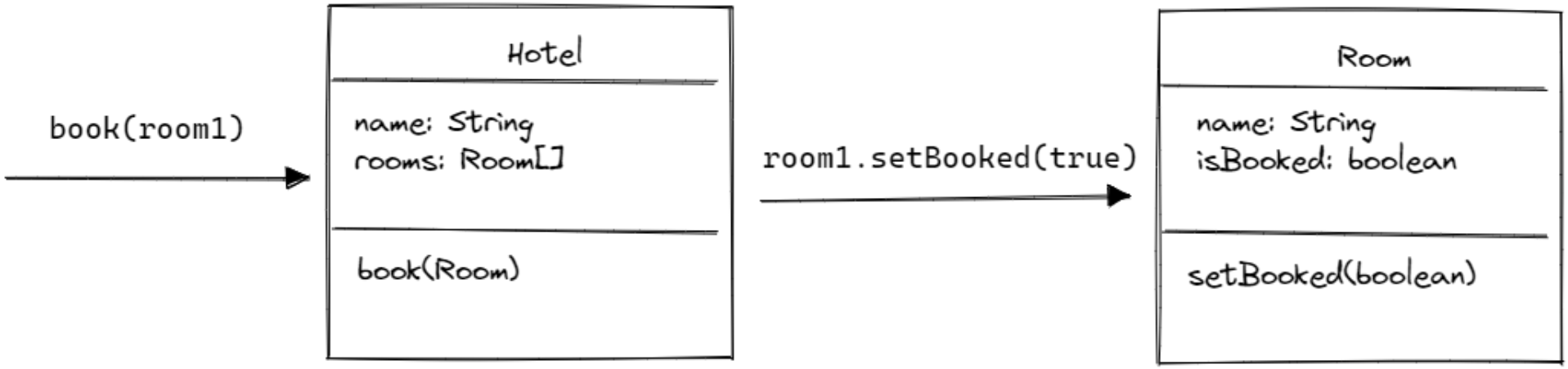
Concept: Polymorphism

- Means "Having many forms"
- Depending on context, an object can be used as any of its types
- Person can also be:
    - Trainee
    - Partner

- Employee
- Brother/Sister

Concept: Message Passing / Method Calling

- Objects communicate to build behaviour
- Java calls this Method Calling



Summary: Concepts of OOP

- Classes
- Objects / Instances
- Data Abstraction
- Inheritance

- Polymorphism
- Encapsulation
- Message Passing / Method Calling

Assignment-1: Jon’s Magic Box

Assignment-2: Another type of Employee

Assignment-3: Painting the Shape

Assignment-4: Identify Classes

- Identify classes to describe a real world entity
- Two cases in Syllabus:
    - Case 1: A Car
    - Case 2: A School