Introduction to Systems Programming

Introduction

2025 - Simón Gutiérrez Brida *(Based on material by Dr. Nazareno Aguirre)*

Lecturer Introduction

Undergraduate in Computer Science

PhD in Computer Science

Researcher on Software Engineering and Formal Methods

Lecturer Introduction

I focus on improving software quality. I approach this problem by researching automatic program/model repair, automatic test generation and test evaluation; and improving automatically generated test inputs.

Lecturer Introduction

I have delved in the darkest corners of Java and will try to make a bash script anytime I can.

Course Introduction

Lecture days, exams, and office hours

- **Lectures:**
  - Mondays 16:15 - 18:15 E2-201 (SC)
- **Tutorials:**
  - **Group A:**
    - Thursdays 8:00 - 10:00 E2-201 (SC)
  - **Group B:**
    - Fridays 8:00 - 10:00 E2-201 (SC)

- **Final exams:**
  - Exam A, June 26Th 14:00 - 17:00
  - Exam B, July 14th 14:00 - 17:00

- **Office hours:**
  TBA (Office TBA)

- **Email address:**
  simon.brida@gtiit.edu.cn
  - I prefer email over moodle messages.
  - The course's moodle forum is also a good option.

You can always come to my office, if I'm there and I'm free I can give office hours.

You can also send me an email asking for office hours and I will try to find the best available time.

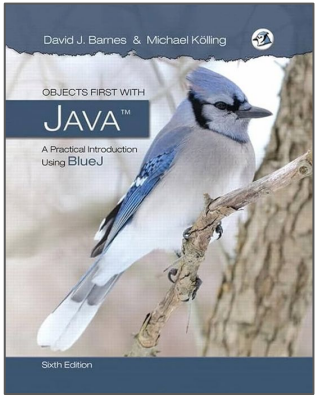
Course Introduction

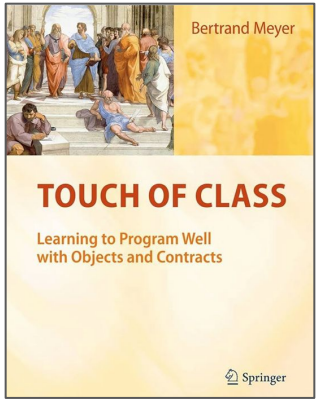
One of the main tool for this course, BlueJ, works fine in Windows, Linux, and macOS. But later on the course we will use other tools that are already available on Linux and macOS, although can be installed (with a bit of work) in Windows. In my case I'll be using Ubuntu 22.04.

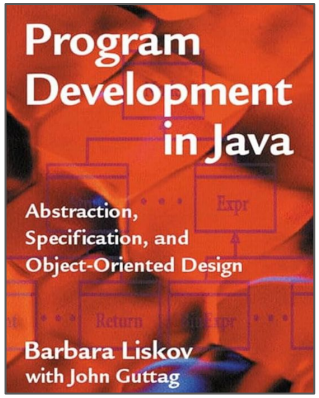
I will also provide online resources and can help you installing Ubuntu and any tool we will use.

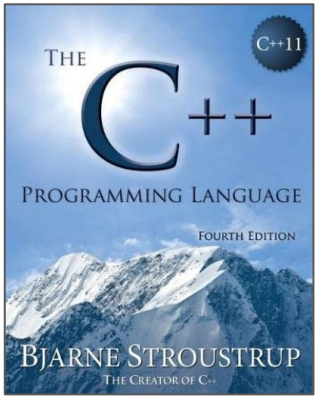
Except when using BlueJ, we do not require an IDE (Integrated Development Environment) since our code will be relatively small. Even though an IDE usually is very helpful, is important that we learn what happens "Under the hood" instead of relying on just "pressing buttons" that we don't really understand what they do.

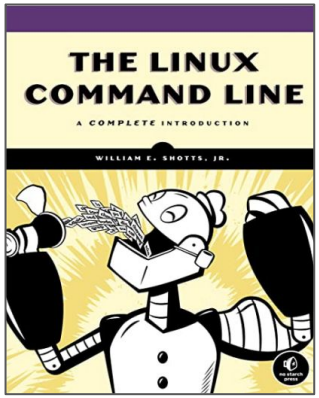
Course Introduction











Course Introduction

What is this course about?

We will learn about Object-oriented programming and its fundamentals; we will also learn fundamental concepts of software design, construction and maintenance; we will apply these concepts to the construction of quality software. The course will end with a short comparison of Java, Python, and C++ to understand how these languages would allow us to apply Object-oriented programming, and how they differ. *(*)*

The weekly plan and more information is posted in the course's moodle, although it is subject to change.

(*) This is a very simplified version of what we will see.

Evaluation and grades

- Assignments:
  - The course will have several graded assignments.
  - Assignments will have a hard deadlines, no submissions will be accepted after the deadline.
  - Submissions will always be through moodle, no email submissions will be accepted.
  - If an assignment is not submitted, it will have a grade of zero.
  - Assignments will be made in groups of 2 students.
- Finals: final exams will include practical and theoretical questions, they will be “open book”.
- The final grade of the course will be based on the final exam’s grade, and assignments. The grade will be calculated as:
  average(assignments) * 0.3 + FE * 0.7.

Some extra notes

- Anyone can make mistakes, that how we learn.
- If you don’t agree with or don’t understand something I say, please tell me.
- We will use books as tools, this means we are not required to fully read all books, and we are not required to follow the chapters in their order.
- A lot of times there are multiple correct answers for the same question/problem.
- I’ll do my best to update the course’s Moodle page with slides and code examples, please let me know if I forget.
- We should always have a 10 minute break on classes that are 2hs long after 50 or 60 minutes, please remind me if I forget.
- We should end the class 10 minutes early (GTIIT policy), please remind me if I forget.

Course structure

- Object Oriented Programming [in Java].
  - This will represent the bulk of the course.
- Introduction to C++.
- Introduction to Python.
- Comparing Java, Python, and C++.

Object Oriented Programming [in Java]

- Objects and classes
- Object interaction and composition
- Program parametrization
- Aggregations (collections) and their treatment
- Ad hoc data representation
- Class design
- Software applications structure
- Inheritance and abstraction
- Error treatment (exceptions)
- Design by Contract and Software Specification

Introduction to C++

- Managing dynamic memory ourselves.
- Pointers.
- Arguments, by value, by reference.
- Classes.
- Constructors/Destructors/Assignments.
- Templates and Generic Classes.
- Inheritance and Polymorphism
- Generic Algorithms (function objects)

Introduction to Python

- What is an interpreter?
- What is Python?
- Python characteristics, programming in Python.
- Python Environment.

Resources

First of all, here are the main tools we will use. Some can be easily installed in any OS, others will be easier to use in macOS or Linux (I'll be using Ubuntu 22.04), there will be a full list of tools provided in the course's Moodle page.

bluej: *A Java IDE designed for teaching/learning.*
g++: *GNU C++ Compiler (it's basically the same as gcc with different flags).*
Any simple text editor: *There's no need for an IDE outside of bluej, our code will be relatively simple.*
Terminal/Console: *we will do some work that requires a terminal/console.*
man: *This is a tool that everyone should always consider, it allows to read manuals for almost every application installed, it also contains C and C++ manuals.*
Python's interpreter (3.0+)

Important terminology

Program: a set of instructions that can be executed i.e.: the code you write.

Algorithm: a finite set of instructions that can be executed

Executable: a file with a specific format, e.g.: ELF, which contains a compiled program which can be executed.

Process: an instance of an executable that is being executed.

** It's common to see "Program", "Executable", and "Algorithm" used interchangeably.*

Example (Division)

Εὐκλείδης
ca. 350 BC — ca. 250 BC
(Euclides of Alexandria)

Theorem: Given **n** and **d** with **d ≠ 0**, there exist **q** and **r** s.t.:

i. **n = (d × q) + r**
ii. **0 ≤ r < d**

Algorithm: Given **n** and **d** with **d ≠ 0**, find **q** and **r** s.t.:

i. Start with **q in 0**
ii. **Subtract d from n**
    a. **If** the **result is negative, restore n** and **set r as n**
    b. **If** the **result is positive, increase q by 1, got** back **to ii**
iii. **Return q** and r

Example (Division)

```
int divide(int n, int d) {
  int q = 0;
  int r = n;
  while (r >= d) {
    r = r - d;
    q++;
  }
  return q;
}
```

Is this a program?

Example (Division)

```
int divide(int n, int d) {
  int q = 0;
  int r = n;
  while (r >= d) {
    r = r - d;
    q++;
  }
  return q;
}
```

Is this an algorithm?

Coding

THE GOOD,

- Meets requirements
- Flexible and easy to maintain
- Reliable (tested and debugged)
- Portable
- Efficient
- Self documented

Coding

THE GOOD,

- Meets requirements
- Flexible and easy to maintain
- Reliable (tested and debugged)
- Portable
- Efficient
  - *never start by optimizing, only optimize after the software correctly solves the main problem.*
- Self documented

Coding


```c
#include<stdio.h>

#define LENGTH 256

/*
* Takes a path to a file and prints how many lines the file has.
*/
int main(int argc, char ** argv) {
  const char * const fname = argv[1]; //check argc > 1
  FILE * file = fopen(fname, "r"); //check result
  char line[LENGTH];
  // read file line by line
  while(fgets(line, LENGTH, file)) {
    // DO SOMETHING WITH LINE
  }

  // CHECK BETWEEN EOF OR IO FAILURE

  fclose(file);

  return 0;
}
```

Coding

- Breaking one functionality during the development of another.
- Complexity in understanding and maintaining the code.
- Difficult to test feature in isolation
- Rigid (a small change causes a cascade of changes).
- Difficult to reuse common functionalities.

Coding


```
int main (int argc, char ** argv) {
  char *path1; char l1[80]; char l2[80]; char l3[80];
  int fd1 = open(argv[1], O_RDONLY | O_CREAT);
  char c; int i=0;

  while ((read(fd1, &c, 1) == 1 && i < 80) {
    l1[i++]=c;
    if(c=='\n') break;
  }

  while ((read(fd1, &c, 1) == 1 && i < 80) {
    l2[i++]=c;
    if(c=='\n') break;
  }

  while ((read(fd1, &c, 1) == 1 && i < 80) {
    l3[i++]=c;
    if(c=='\n') break;
  }

  ...

}
```

Coding


```
// Dear maintainer:

// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:

// total_hours_wasted_here = 42
```

```
// I dedicate all this code, all my work, to my wife who
// will have to support me, our children, and the dog
// once it gets released into the public.
```

```
// Used to work around Richard being an idiot
```

```
// You may think you know what the following code does.
// You DON'T. TRUST ME.
// Fiddle with it, and youll spend many a sleepless
// night cursing the moment you thought youd be clever
// enough to understand what the code below does.
// Now close this file and go play with something else.
```

Classes and objects

Starting with a figures demo

Classes and Objects

Objects represent entities from the real world, in a specific problem domain, and even “internal” software entities. And classes represent all objects of a specific kind.

Classes and Objects

Objects represent entities from the real world, in a specific problem domain, and even "internal" software entities. And classes represent all objects of a specific kind.

Some examples:

- A **class** representing all bicycles, and a particular red bicycle would be an **object** of that class.

Classes and Objects

Objects represent entities from the real world, in a specific problem domain, and even "internal" software entities. And classes represent all objects of a specific kind.

Some examples:

- A **class** representing all students, and a particular student called "Arthur Dent" with ID number "42" would be an **object** of that class.

Classes and Objects

Objects represent entities from the real world, in a specific problem domain, and even "internal" software entities. And classes represent all objects of a specific kind.

Some examples:

- A **class** representing all possible calculators, and a particular calculator that we are using would be an **object** of that class.

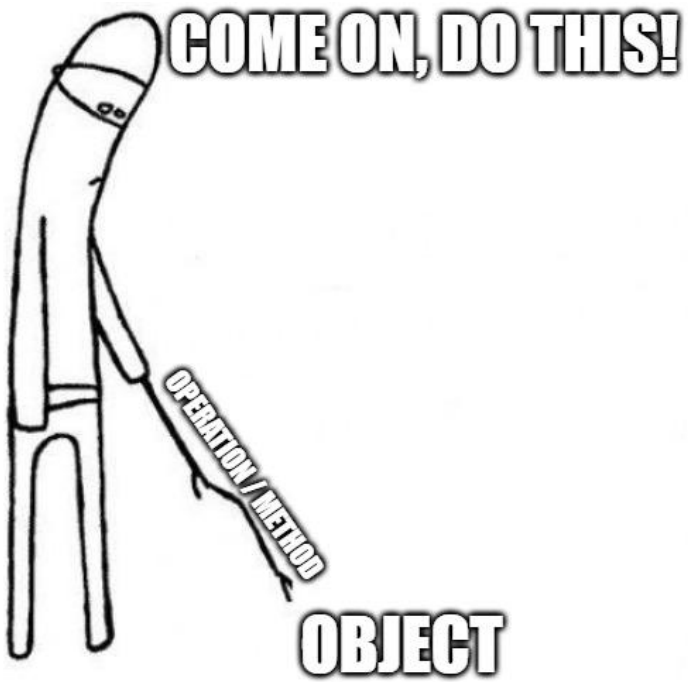
Methods and parameters

Methods and parameters

Once we have an object of a class, we need to be able to do something with it, if not, then it's completely useless.

How we do it?

Methods and parameters

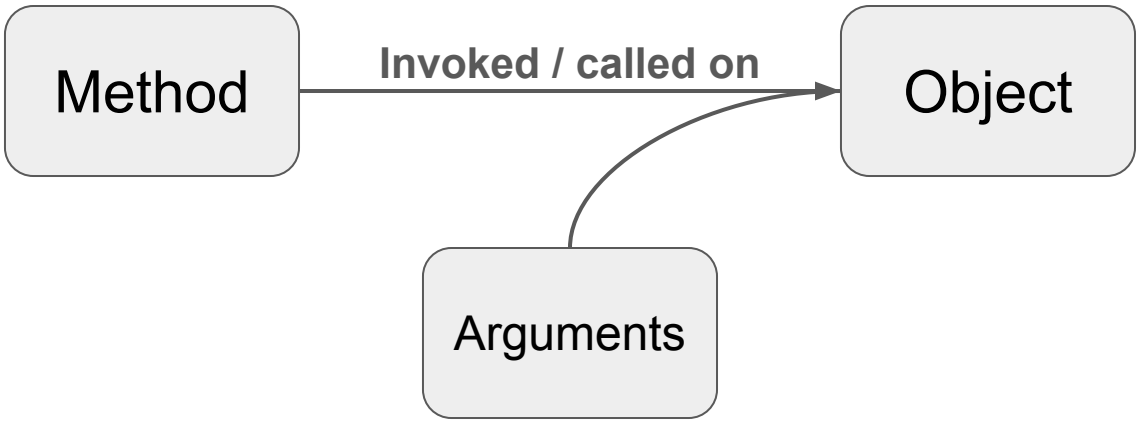


Once we have an object of a class, we need to be able to do something with it, if not, then it's completely useless.

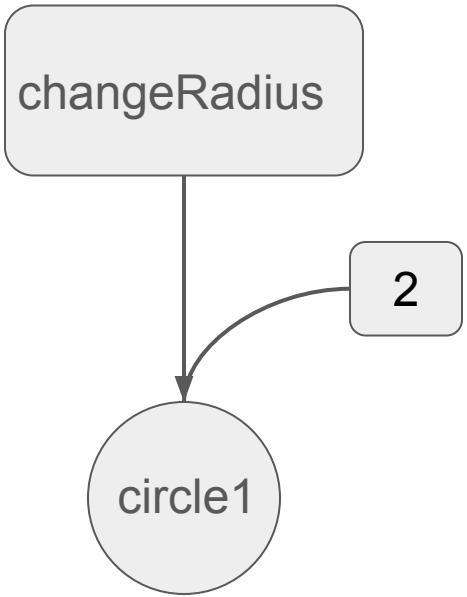
We apply and operation to the object, most commonly known as methods.

Methods and parameters

In many cases we want to apply an operation/method to an object which involves one or more arguments.



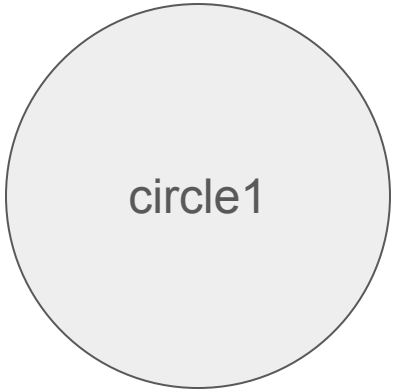
Methods and parameters



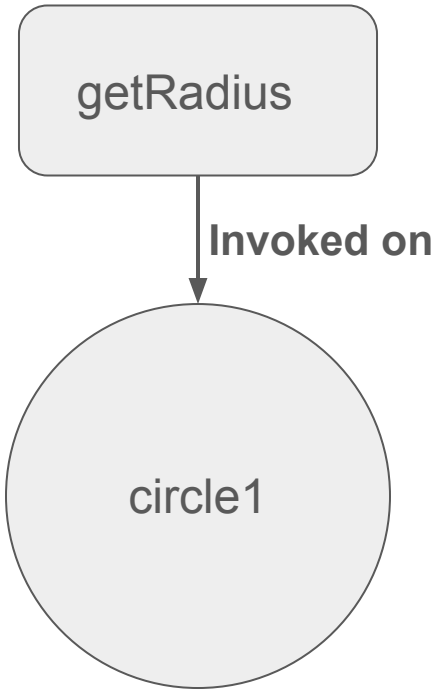
In many cases we want to apply an operation/method to an object which involves one or more arguments.

Methods and parameters

Methods can also return a value

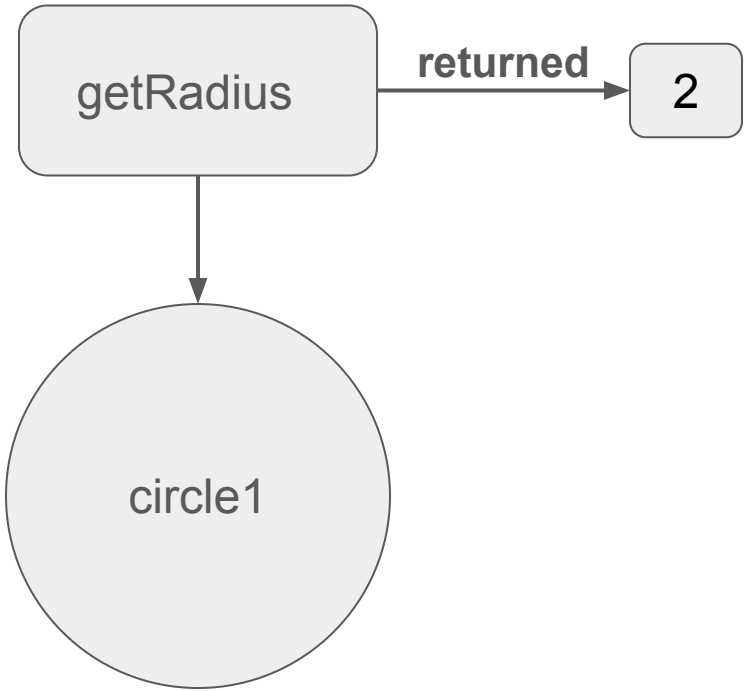


Methods and parameters



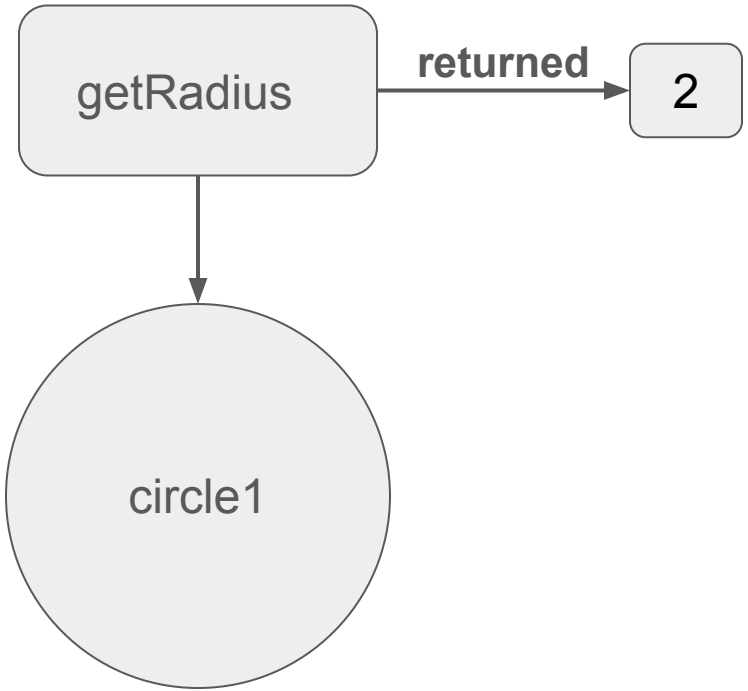
Methods can also return a value

Methods and parameters



Methods can also return a value

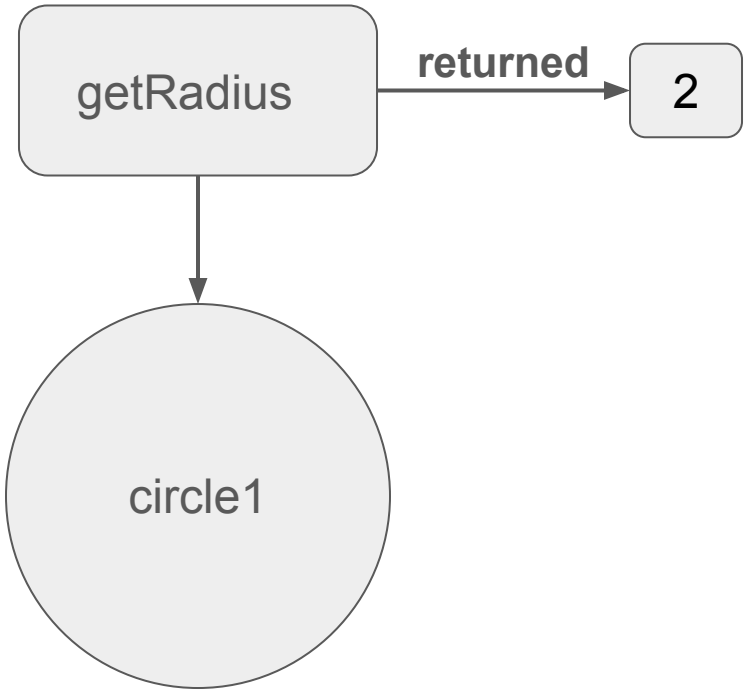
Methods and parameters



Methods can also return a value

Where are methods defined?

Methods and parameters



Methods can also return a value

Where are methods defined?

Inside classes
(we will talk about it later)

Data types

A Data Type defines a set of values and operations that can be applied to those values.

Examples:

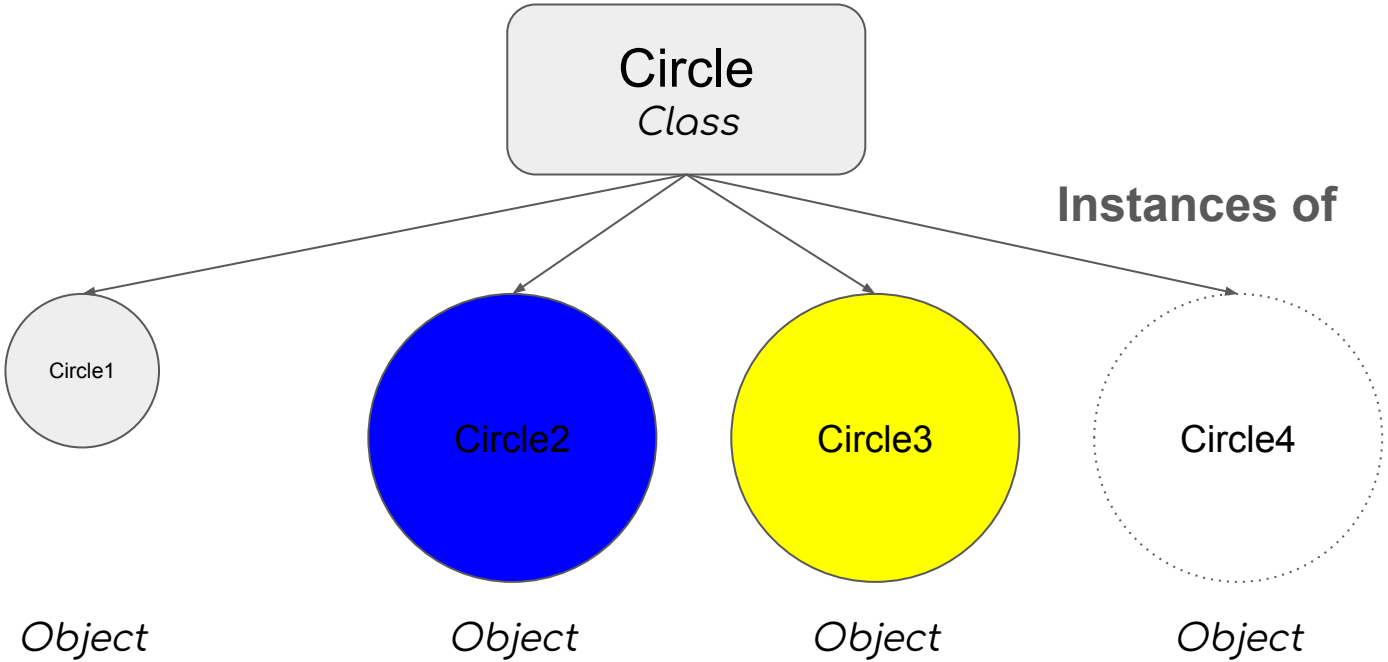
int: set of all integer values.

boolean: set of boolean values (truth values).

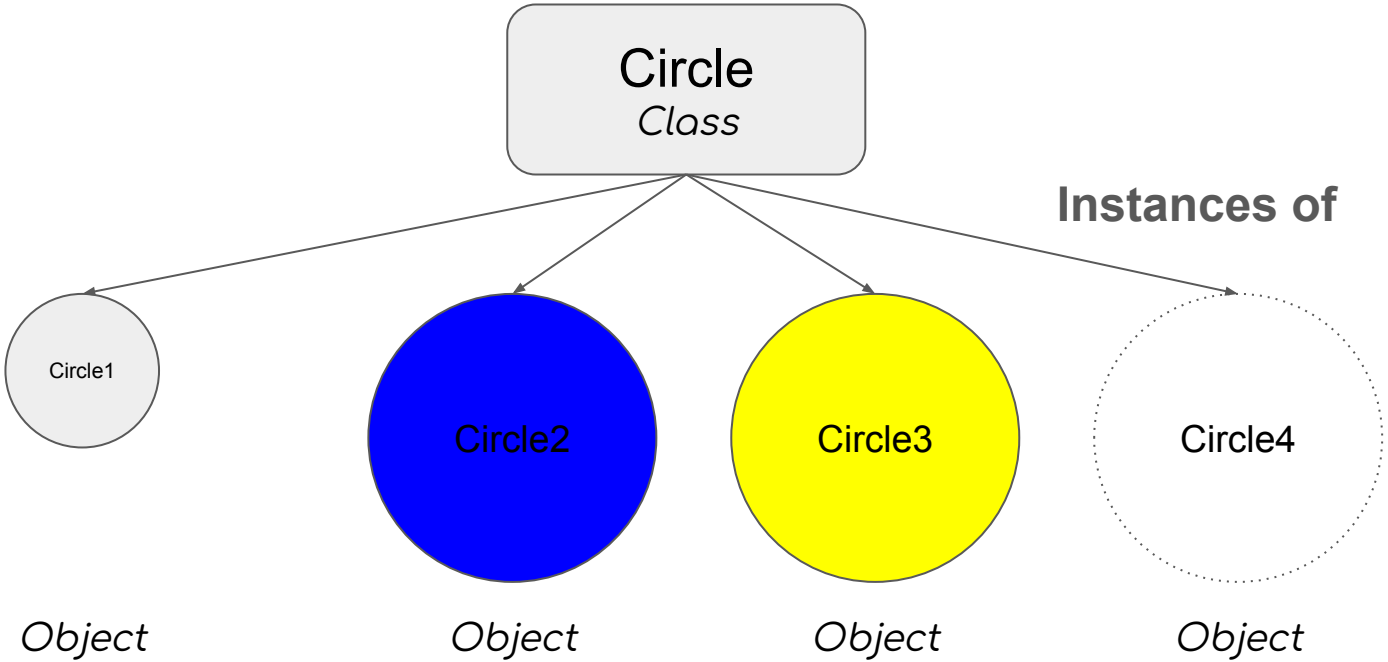
floats: set of real values.

char: set of all character values (ASCII).

State of an Object

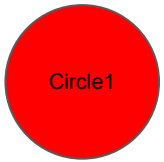


State of an Object

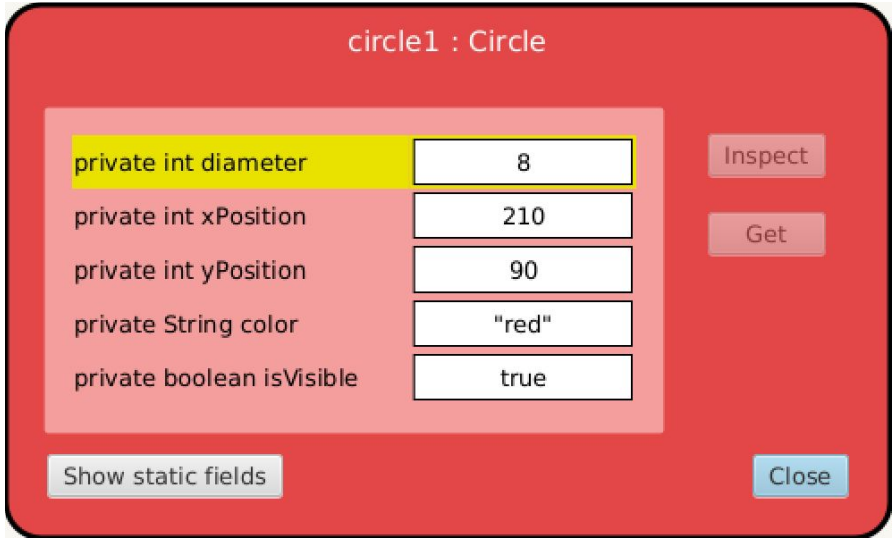


Each object has attributes/fields
Mutable values that define relevant information of the objects

State of an Object

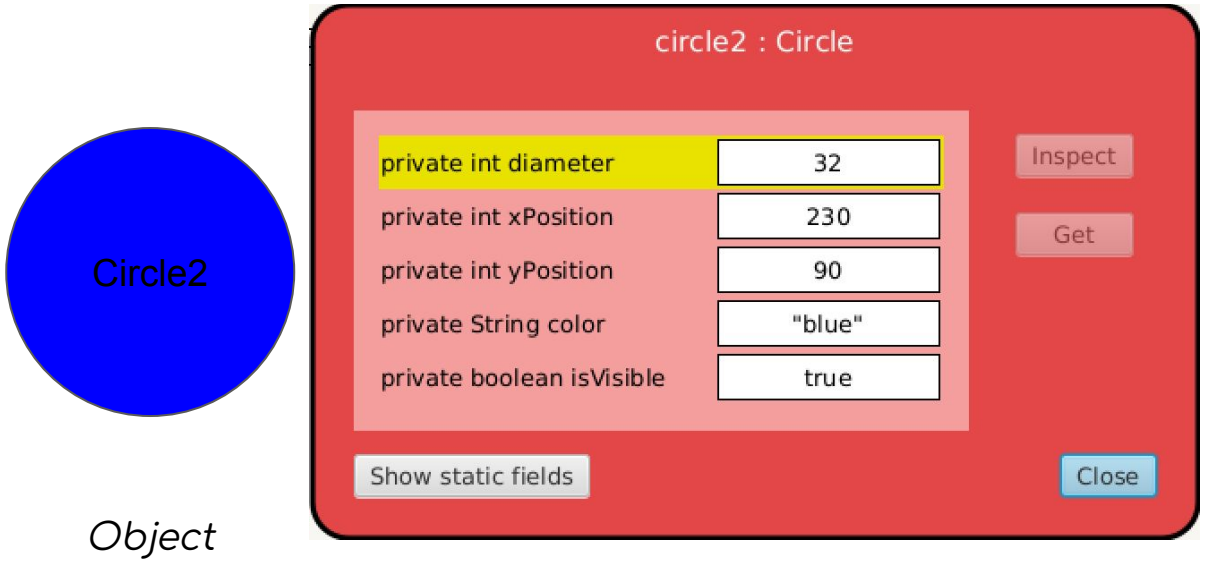


Object



Each object has attributes/fields
Mutable values that define relevant information of the objects

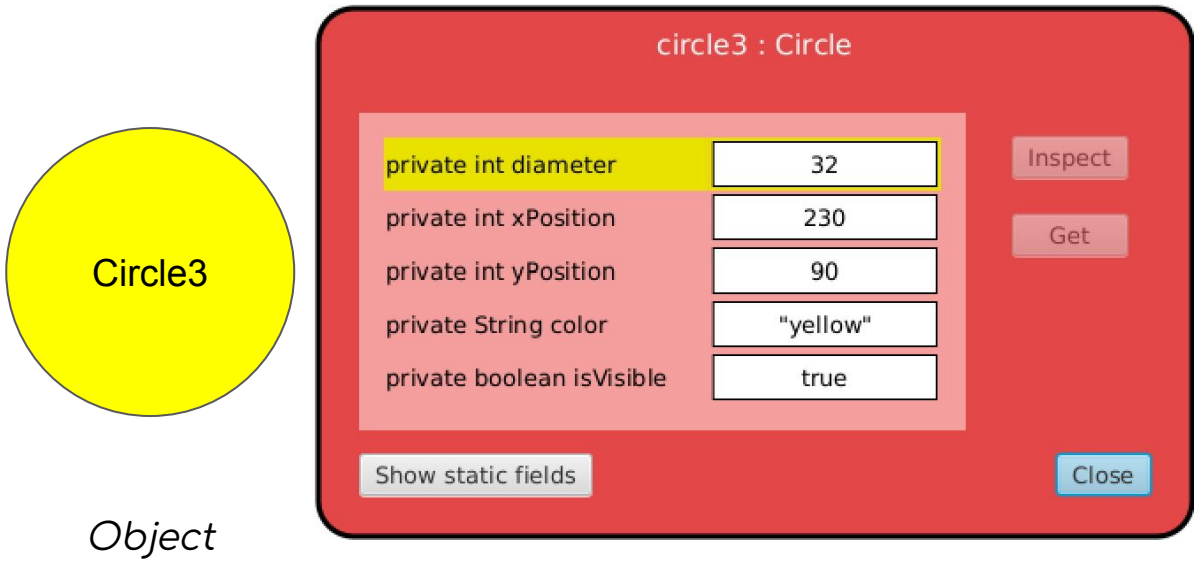
State of an Object



Object

Each object has attributes/fields
Mutable values that define relevant information of the objects

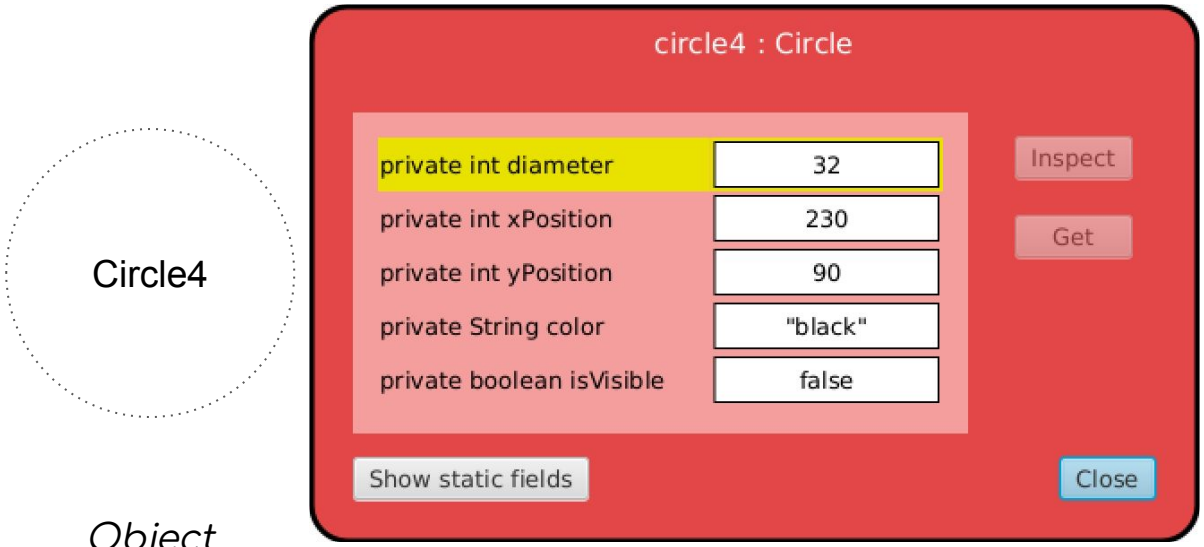
State of an Object



Object

Each object has attributes/fields
Mutable values that define relevant information of the objects

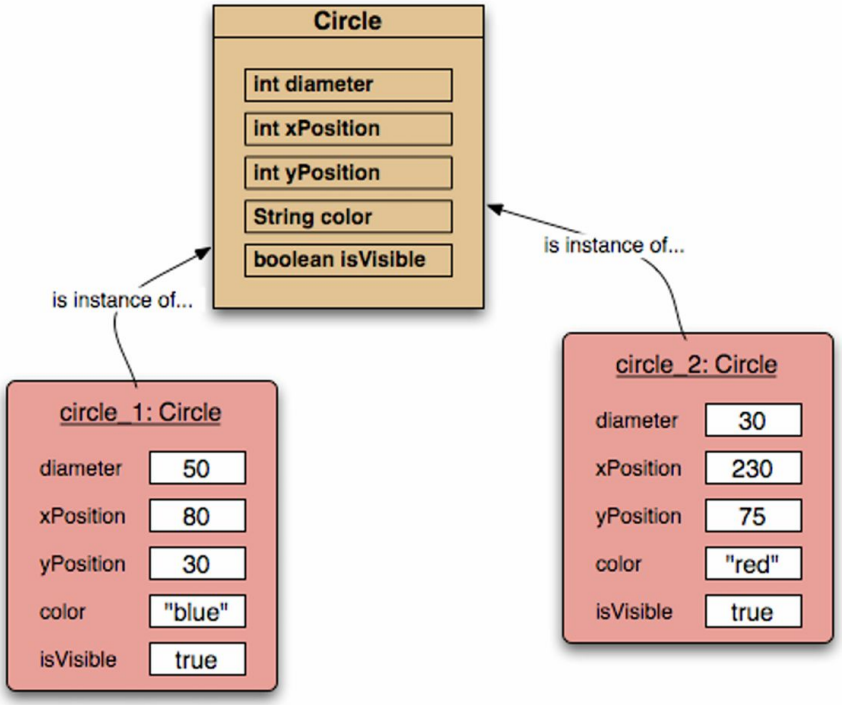
State of an Object



Object

Each object has attributes/fields
Mutable values that define relevant information of the objects

State of an Object



A class defined which attributes/fields will objects of that class will have

Source Code

Classes are a way of representing and organizing computer programs.

Source Code

Classes are a way of representing and organizing computer programs.

Classes are written in source code (Java code), and defines how objects will be represented and how they will behave..

Source Code

Classes are a way of representing and organizing computer programs.

Classes are written in source code (Java code), and defines how objects will be represented and how they will behave..

Java code is compiled into a low-level language called Bytecode that will be executed by a Java Virtual Machine.

Class Definitions

Starting with a ticket machine demo

Roadmap of topics

- Fields (also known as attributes)
- Constructors
- Methods
- Parameters
- Assignment

Ticket Machine - an external view

- By interacting with an object, we can have clues of what is the behavior of the object.
- By examining the internal definition of an object, we can have an insight about how such behavior is provided, or implemented.
- All Java classes have a consistent internal structure.

Structure of a class

```
public class ClassName {
    Fields
    Constructors
    Methods
}
```

External definition of a class
(header of a class)

Internal definition of a class
(contents of a class)

Fields

- Class fields define the internal values an object will hold.
- In BlueJ, one can use the Inspect option to see the fields of an object.
- Class fields define the state space of an object.

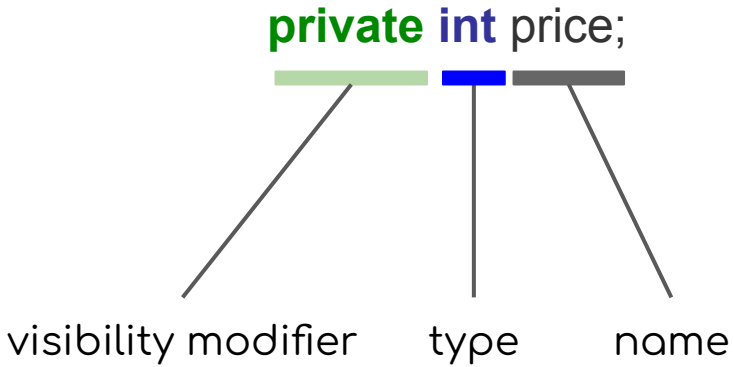
```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    //Additional details omitted.

}
```

Fields

- Class fields define the internal values an object will hold.
- In BlueJ, one can use the Inspect option to see the fields of an object.
- Class fields define the state space of an object.



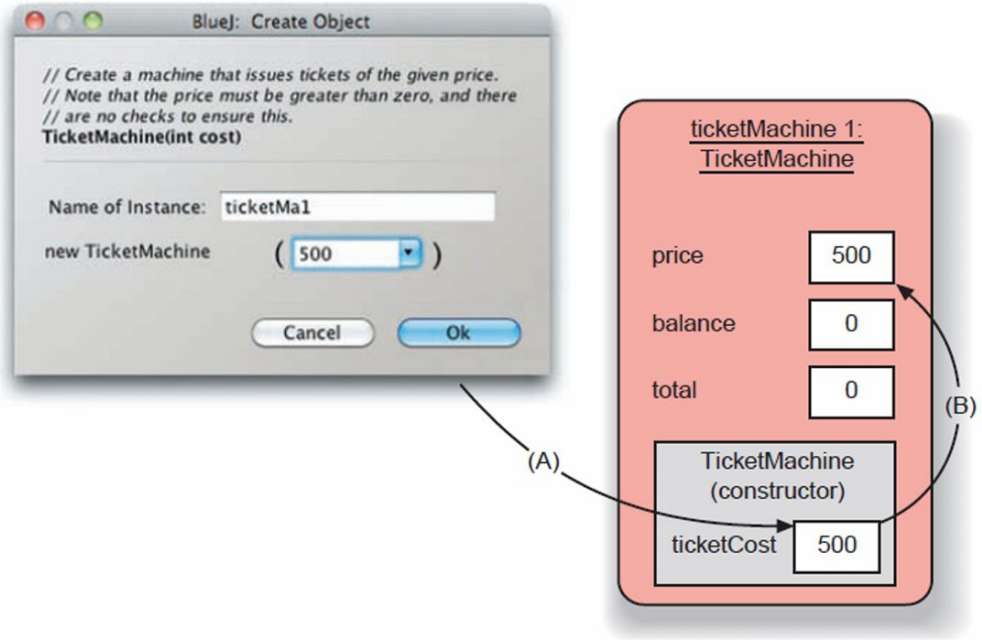
Constructors

- Class constructors are responsible of initializing objects
- They have the same name as the name of the class.
- They take care of storing initial values into fields.
- Frequently, constructors receive external values (parameters) to perform the initialization

```
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

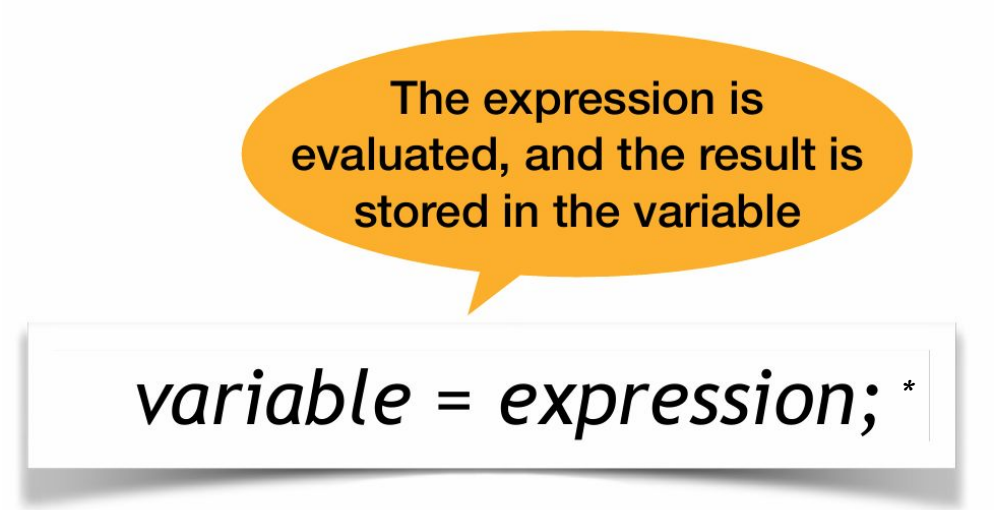
Passing data via parameters

- Parameters are a kind of variable
    - They take care of passing data from the "client" of the method to the method itself.
    - As opposed to fields, parameters are only accessible within the method's body.



Assignment

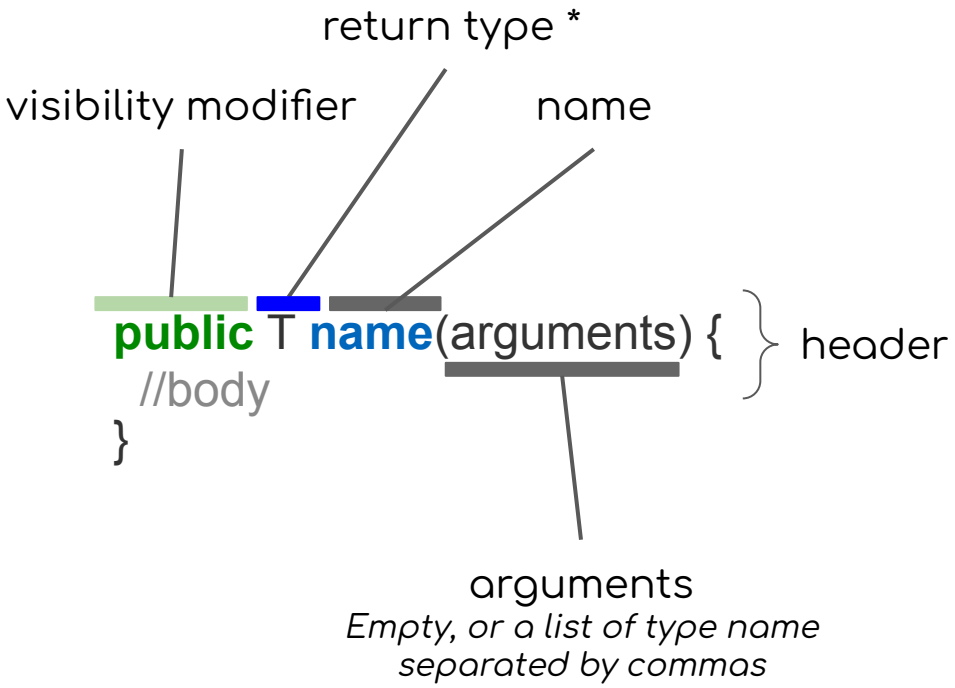
- Values are stored in fields (and other kinds of variables) using assignment statements.
  - A variable can hold a unique value.
  - The previous value is lost when variable is assigned to.
  - E.g.: price = ticketCost;



(*) The type of the expression, and the type of the variable, must be compatible!

Methods

- The methods of a class implement the behavior of the objects of the class.
  - Methods have a structure, consisting of a header and a body.
  - The header defines the signature of the method.
  - The body comprises the sentences that implement the method.



*(*) If it's not void, the method **must** return a value.*

Methods (Command - Query separation)

A method either represents a command which changes the state of the object in which is invoked (*). Or represents a query, which only returns data associated with the object in which has been invoked.

(*) Printing is considered as a command.

```
public void changePrice(int newPrice) {
  price = newPrice;
}
```

```
public int getPrice() {
  return price;
}
```

Quick quiz!

```
public class CokeMachine
{
  private price;

  public CokeMachine()
  {
     price = 300
  }

  public int getPrice
  {
     return Price;
  }
```

(There's 5 errors!)

Analyzing the naive ticket machine

- What issues we can detect from the initial implementation?
- How we could improve on it?