Introduction to Systems Programming

Features of classes and objects

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

In the last chapter …

An <u>object</u> is an instance of a particular <u>class</u>, which represents a value (with related operation) for specific problem domain.

A <u>class</u>, defines all possible <u>object</u> with a particular collection of data and operations.

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.

In the last chapter …

A class defines the data (Fields), Methods/operations, and Constructors.

Fields determine the state space of all class' objects; Methods the operation that can be invoked on a particular object; and constructors define how new objects are created.

Java code

Class structure

```
public class ClassName {

  //CLASS MEMBERS

}
```

Field structure

```
private Type fieldName;

private type fieldName;
```

Java code

Class structure

```
public class ClassName {

  //CLASS MEMBERS

}
```

Constructor structure

```
public ClassName(/*ARGUMENTS*/) {
  //BODY
}
```

Java code

Class structure

```
public class ClassName {

  //CLASS MEMBERS

}
```

Method structure

```
public Type methodName(/*ARGUMENTS*/) {
  //BODY
}


public type methodName(/*ARGUMENTS*/) {
  //BODY
}
```

Java code, arguments

Arguments: a list (possibly empty), of elements with the format

type|Type **name**

separated by a comma.

```
int a
String author
char c
```

Java code, body

body:
- statement;
- statement; body
- { body }

statement:
- assignment
- variable declaration
- return
- method call

Java code, body

assignment: variable = expression

variable declaration:

```
Type variableName = expression
type variableName = expression
```

return:

```
return expression
```

method call:

```
obj.methodName(/*arguments*/)
```

Java code, expressions

An expression is a “string” of Java code that can be evaluated into a value, some examples:

- A value (a string, an int, a char, an object, etc)
- A unary expression (!true, i++, --i)
- A binary expression, i.e.: expression binary_op expression (3 + 1, a + 1, “Hello” + “ ” + “World!”, 4 * 3)
- A method call (for a method that returns a value)
- A constructor call (new ClassName(/*arguments*/))

What is an object in OOP?

It's a software notion: a “machine” known through the operations it admits or supports.

We can consider three kinds of objects:

- Physical objects: They reflect material objects in the world or system being modeled by software, e.g., a bicycle, a car, a book
- Abstract objects: They describe abstract notions from the world or system being modeled, e.g., a plan or schedule, an appointment
- Software objects: They represent notions that are pure “internal” software abstractions, e.g., an iterator, an array, a linked list

A primary aspect of object oriented programming is its modeling ability: it allows one to connect in a relatively natural way software objects with entities from the problem domain (e.g., real world entities).

Objects and Classes

Each object belongs to a specific class, which defines the information or data that the objects hold, and the operations that are applicable to those objects. The object's data and operations are collectively called features.

Objects and Classes, examples

A class Triangle, that represents all triangles in a 2D plane with a specific width and height, a specific color, position (x, y), and if it’s visible or not in the plane.

A class City, representing all cities, with a particular name, habitants, and country.

A class Book, representing all books with a particular title, author, pages, and summary.

Classes

A class is a description of all possible runtime objects, their state space, and operations that can be invoked on those objects.

A <u>class</u> represents a category of things.

An <u>object</u> is one of such things.

Class, instance, generating class

If an object **o** is one of the objects characterized by a class **C**, we say that:

- **o** is an **instance** of **C**
- **C** is the **(generating) class** of **o**

Object vs Classes

Classes only exist as definitions (like a C struct):

- They are defined by the class text
- They describe the features (fields, methods) of all objects associated with the class (all its potential instances)

Objects only exist at runtime (like a variable):

- They are “visible” in the text of a program through names that denote objects at runtime, i.e., variables (square1, myCircle, etc), as the result of an expression (`myCourse.getStudentByID(42)`).

Method invocation

Invoking methods on objects is what allow us to interact with objects and reach a solution to a specific problem.

A method m can only be invoked on an object o of a class c, if and only if,

o *is an instance of* class c, and class c *defines* method m.

Queries and Commands

Methods can be divided into:

- Commands, these methods modify the state of an object, e.g.: changing the size of a triangle.
- Queries, these methods retrieve, or compute, information of an object, e.g.: retrieving the price of a ticket.

The query-command separation principle establish that a method must either be a command and not return any value; or a query and only return a value without modifying the state of the object.

Introduction to Systems Programming

Conditional composition, Variables,
Scope and lifetime.

Going back to the naive ticket machine

Let's explore the example again, trying to replicate how a real ticket machine would work and see what happens.

Going back to the naive ticket machine

Let’s explore the example again, trying to replicate how a real ticket machine would work and see what happens.

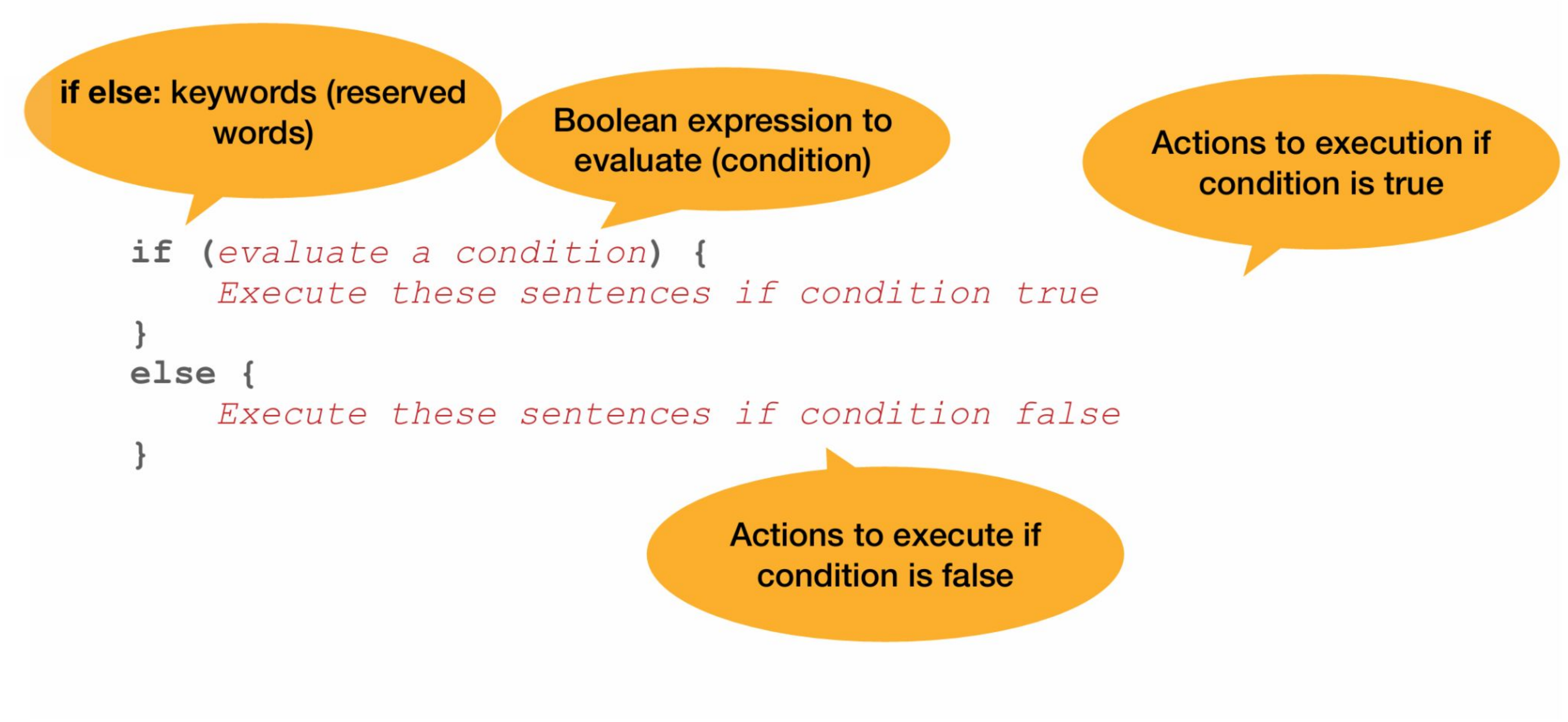
Does the current implementation reflects what is expected of a ticket machine?

Going back to the naive ticket machine

- Their behavior is inadequate, due to various reasons:
    - Inputs are not appropriately validated.
    - Change is not returned to the user.
    - Initialization is not appropriately validated.

- How can we improve the design and implementation of these machines?
    - We will need more sophisticated behavior.

If statement


if else: keywords (reserved words)
Boolean expression to evaluate (condition)
Actions to execution if condition is true
Actions to execute if condition is false

```
if (evaluate a condition) {
    Execute these sentences if condition true
}
else {
    Execute these sentences if condition false
}
```

Java code, body, updated

body:
- statement;
- statement; body
- { body }

statement:
- assignment
- variable declaration
- return
- method call
- If statement
  (does not need a ; at the end)

Java code, body, updated

assignment: variable = expression

variable declaration:

```
Type variableName = expression
type variableName = expression
```

return:

```
return expression
```

method call:

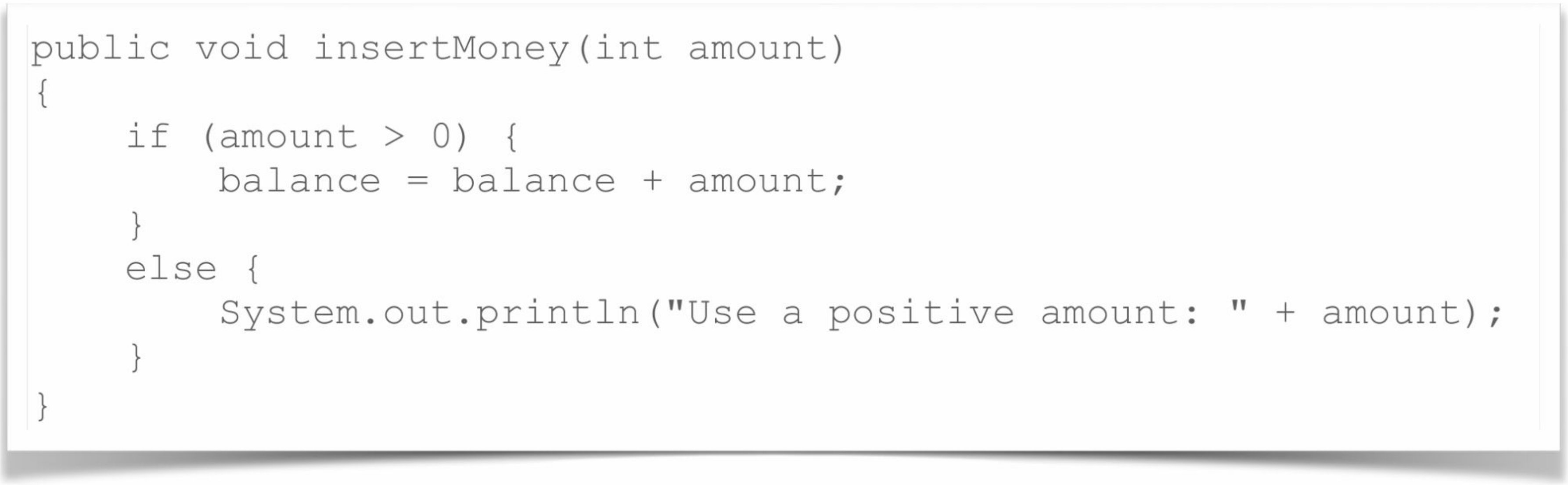
```
obj.methodName(/*arguments*/)
```

If statement:

```
if(condition) {
  //BODY
}

if(condition) {
  //THEN-BODY
} else {
  //ELSE-BODY
}
```

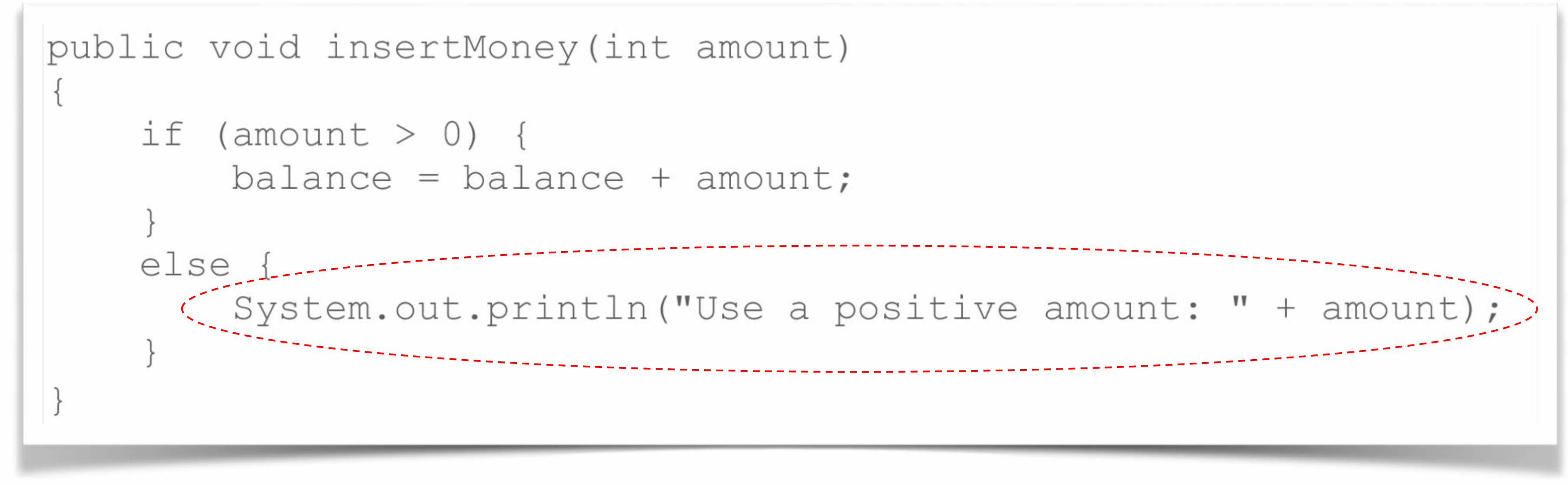
The condition must be a boolean expression

One example of if-statement in the naive ticket machine



```
public void insertMoney(int amount)
{
    if (amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " + amount);
    }
}
```

One example of if-statement in the naive ticket machine



```
public void insertMoney(int amount)
{
    if (amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " + amount);
    }
}
```

Later we will see that this is NOT a good practice!
Can you guess why it is not?

How could we refund balance back to the user?

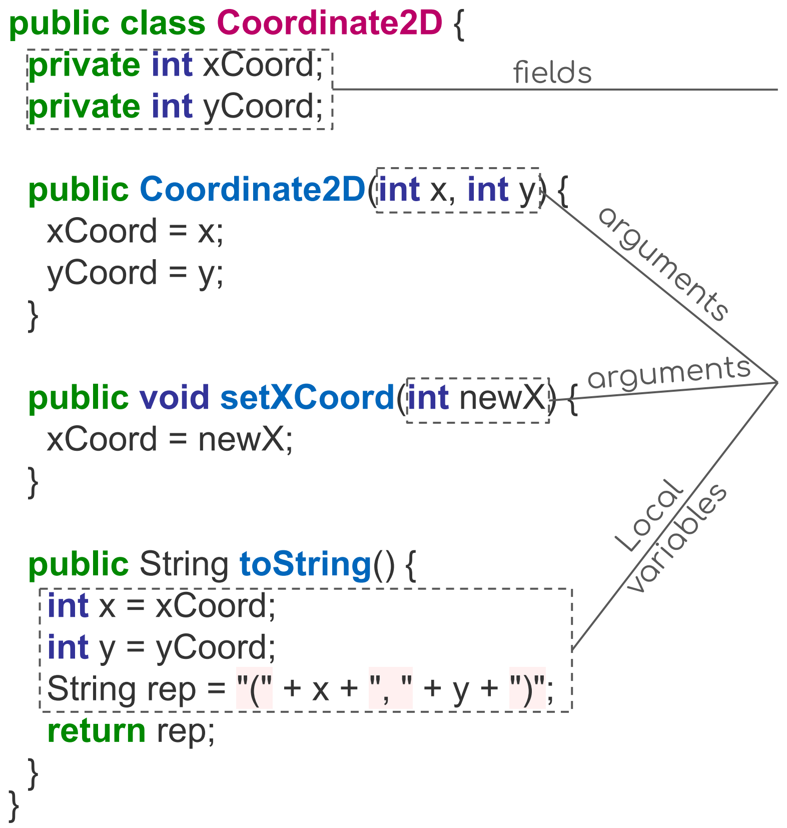
Let’s code for a bit

Variables

Variables store values during their lifecycle, this is determined by their scope.

Let’s see an example!

Variables



```
public class Coordinate2D {
  private int xCoord;
  private int yCoord;

  public Coordinate2D(int x, int y) {
    xCoord = x;
    yCoord = y;
  }

  public void setXCoord(int newX) {
    xCoord = newX;
  }

  public String toString() {
    int x = xCoord;
    int y = yCoord;
    String rep = "(" + x + ", " + y + ")";
    return rep;
  }
}
```

Their scope is the whole class, their lifecycle is the lifecycle of the object.

Their scope is the method/constructor, their lifecycle is the execution of the method/constructor.

The scope of a variable/field is the inside of the block where it was defined.

Introduction to Systems Programming

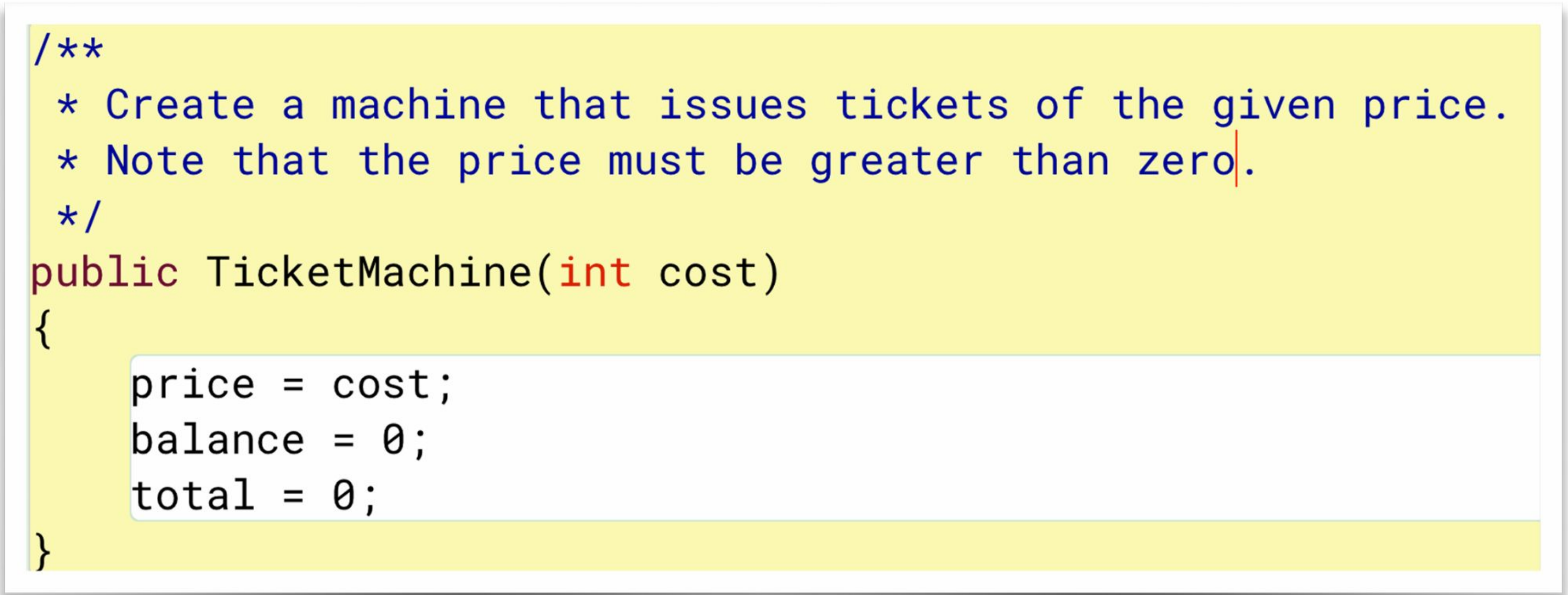
Preconditions. Object interaction.

Precondition

It is a condition that must be met before starting the execution of a method, or constructor.

- It is an obligation of the invoker (the client of the method) to ensure that the precondition holds, before calling the method.
- If the precondition does not hold, the method (and consequently the object that contains it) may behave in an incorrect way.
- Enforcing the satisfaction of preconditions protects the state of objects.
- It supports/enable encapsulation.

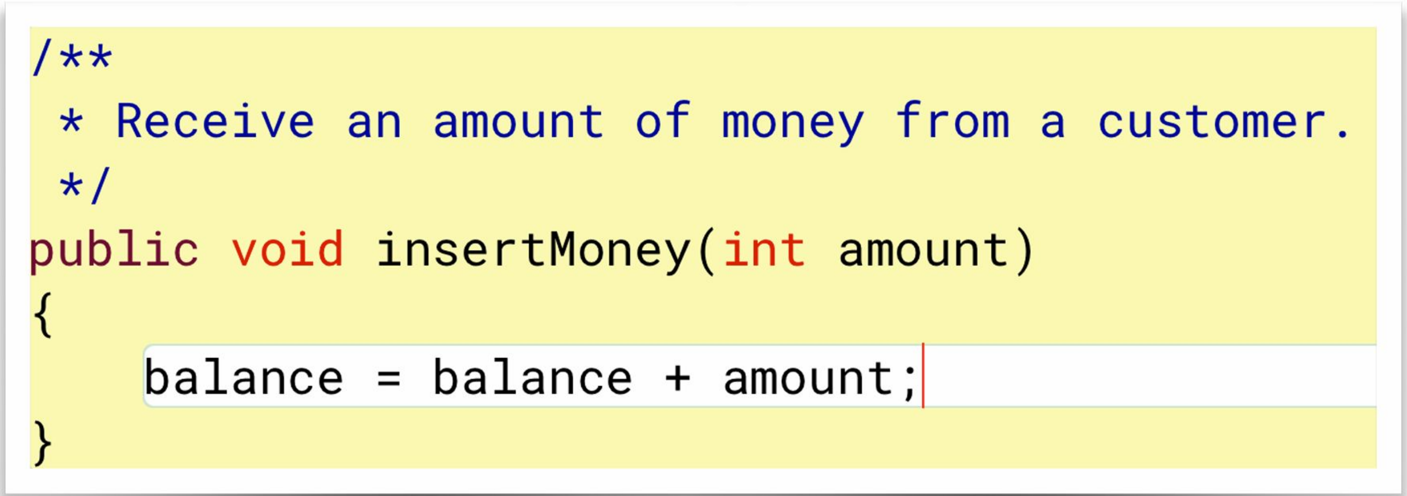
Precondition - Example



```
/**
 * Create a machine that issues tickets of the given price.
 * Note that the price must be greater than zero.
 */
public TicketMachine(int cost)
{
    price = cost;
    balance = 0;
    total = 0;
}
```

Precondition: cost > 0

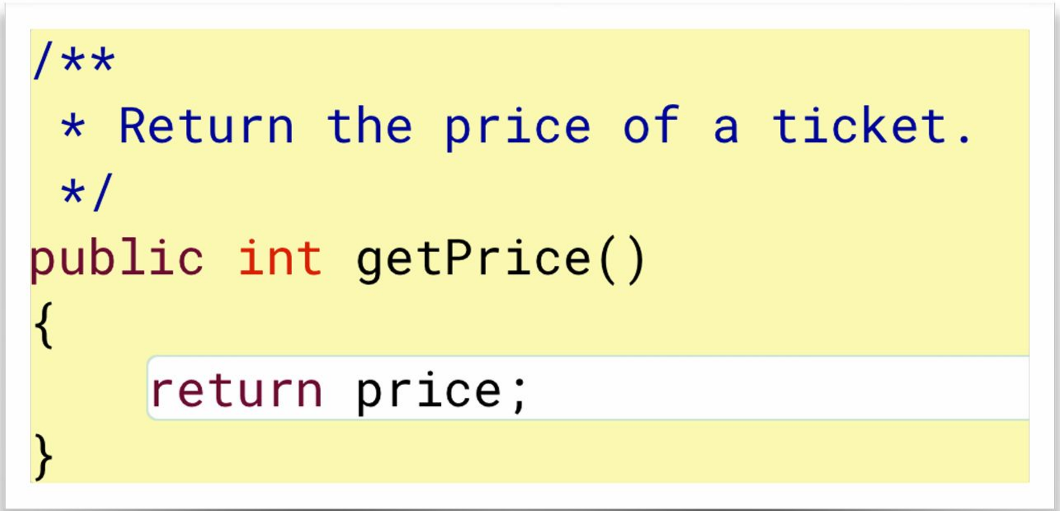
Precondition - Example



```
/**
 * Receive an amount of money from a customer.
 */
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

Precondition: amount > 0

Precondition - Example

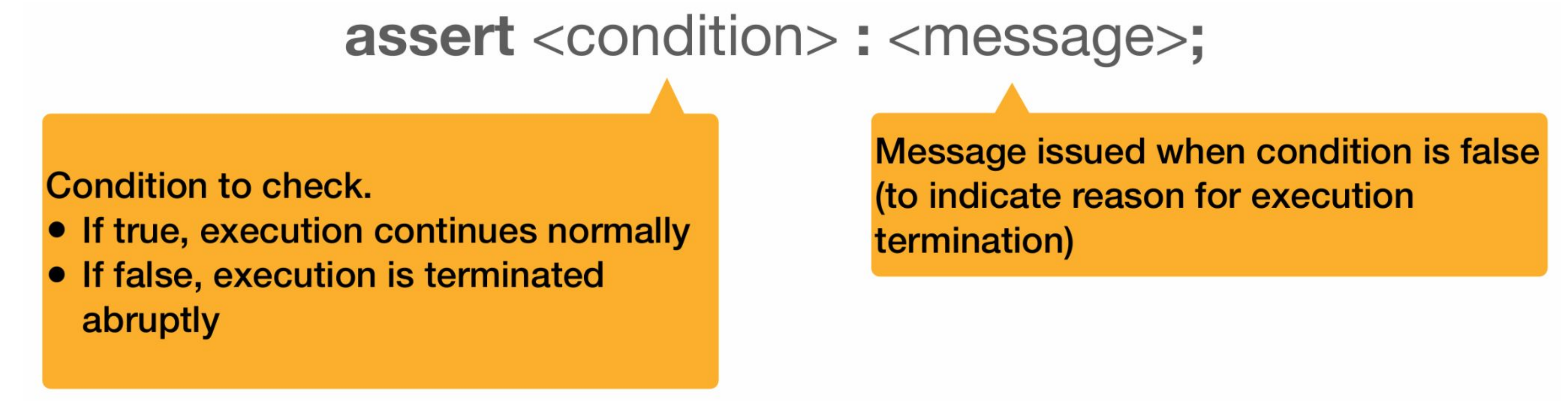


```
/**
 * Return the price of a ticket.
 */
public int getPrice()
{
    return price;
}
```

Precondition: true

Assertions

Java supports assertions, which will allow us to easily implement precondition checking. If the condition of an assertion is false, the program will stop, and an error message will be shown.

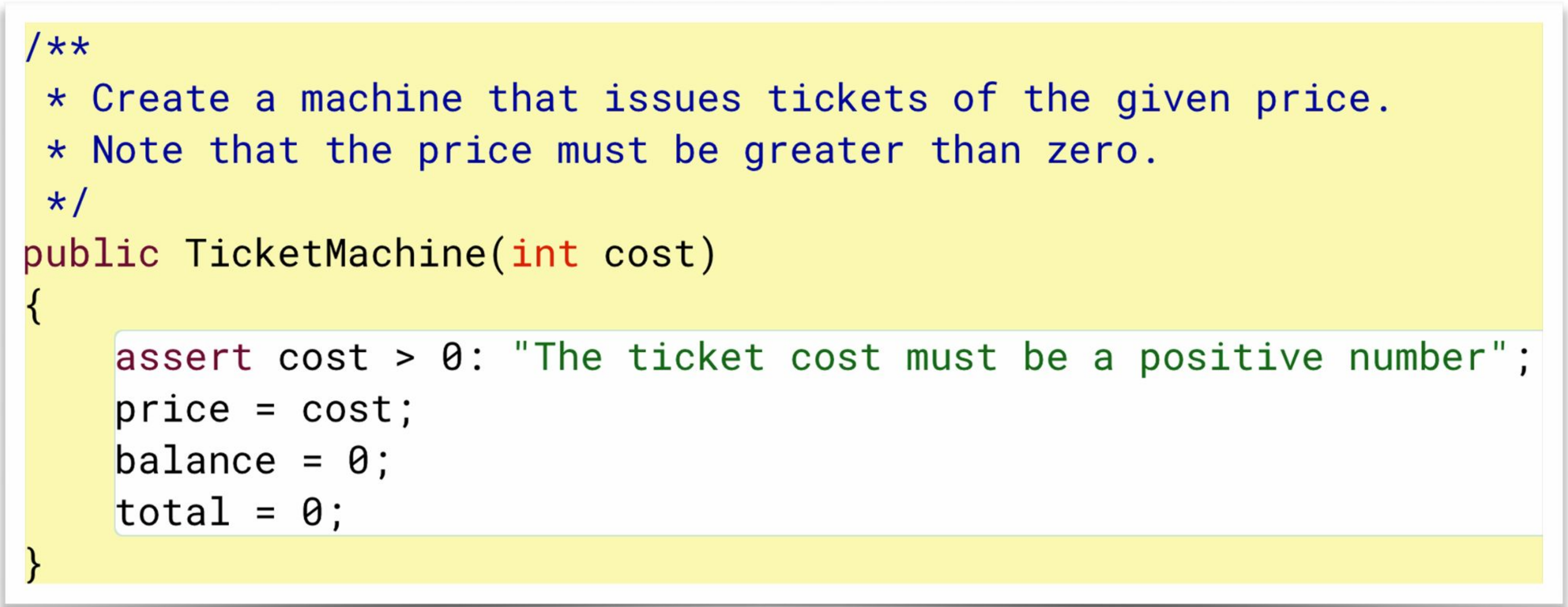


assert <condition> **:** <message>**;**

Condition to check.
- If true, execution continues normally
- If false, execution is terminated abruptly

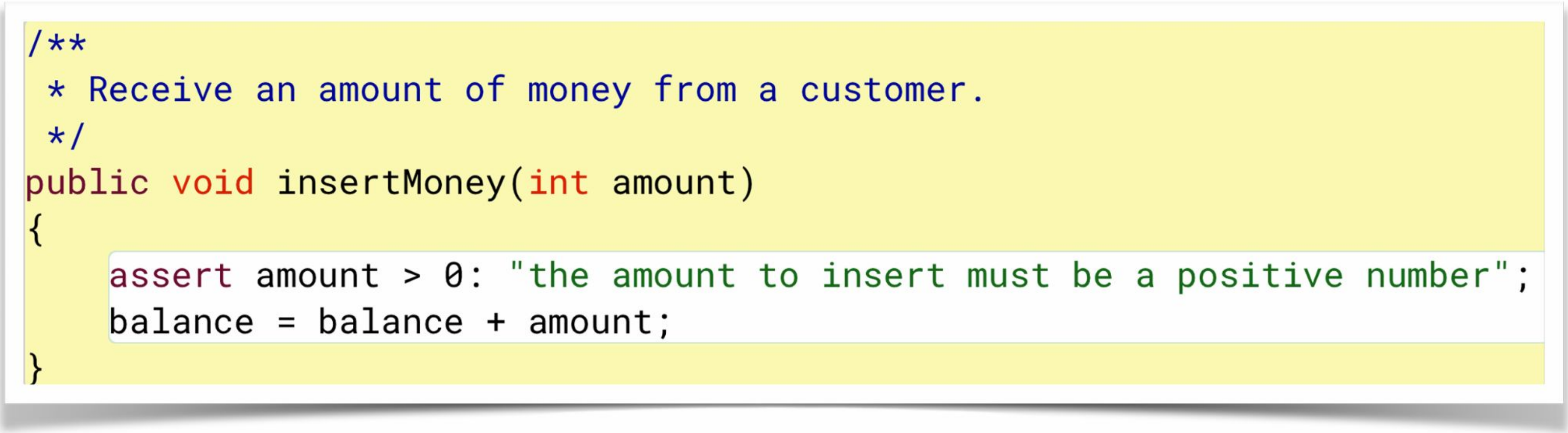
Message issued when condition is false (to indicate reason for execution termination)

Assertions - Example



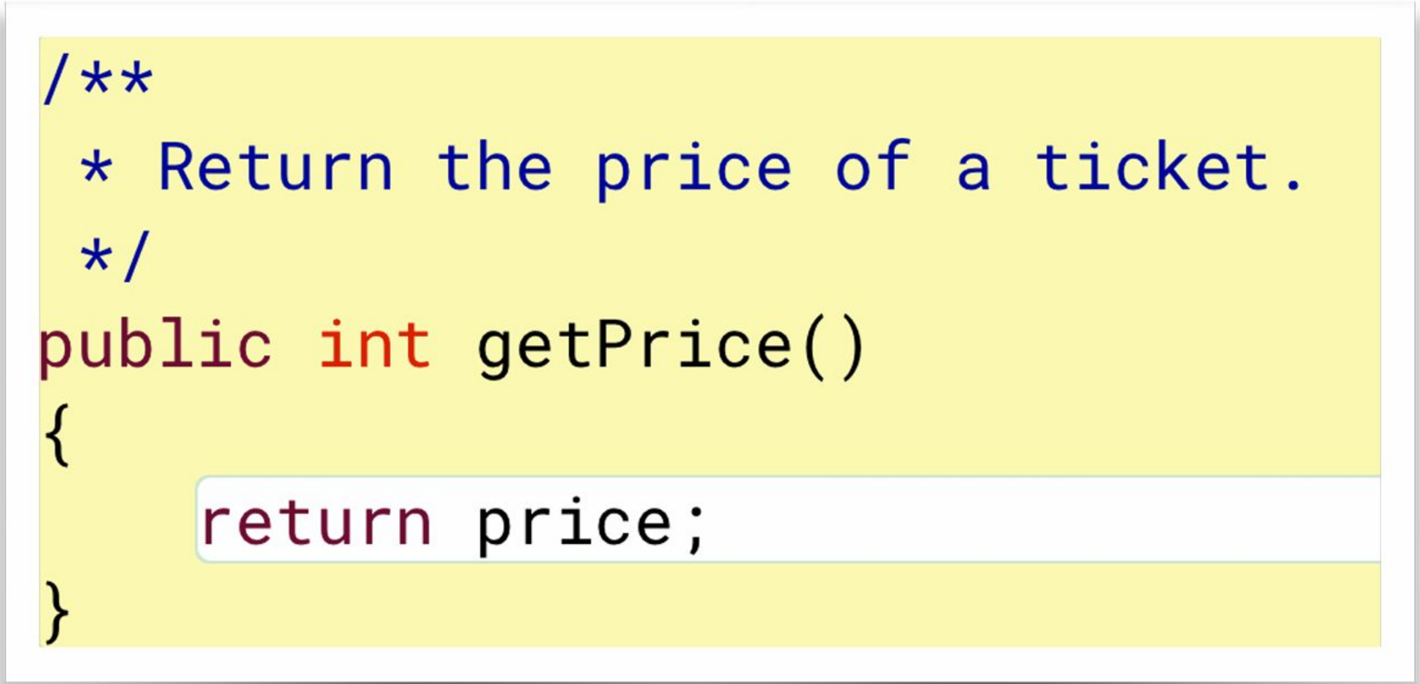
```java
/**
 * Create a machine that issues tickets of the given price.
 * Note that the price must be greater than zero.
 */
public TicketMachine(int cost)
{
    assert cost > 0: "The ticket cost must be a positive number";
    price = cost;
    balance = 0;
    total = 0;
}
```

Assertions - Example



```java
/**
 * Receive an amount of money from a customer.
 */
public void insertMoney(int amount)
{
    assert amount > 0: "the amount to insert must be a positive number";
    balance = balance + amount;
}
```

Assertions - Example



```
/**
 * Return the price of a ticket.
 */
public int getPrice()
{
    return price;
}
```

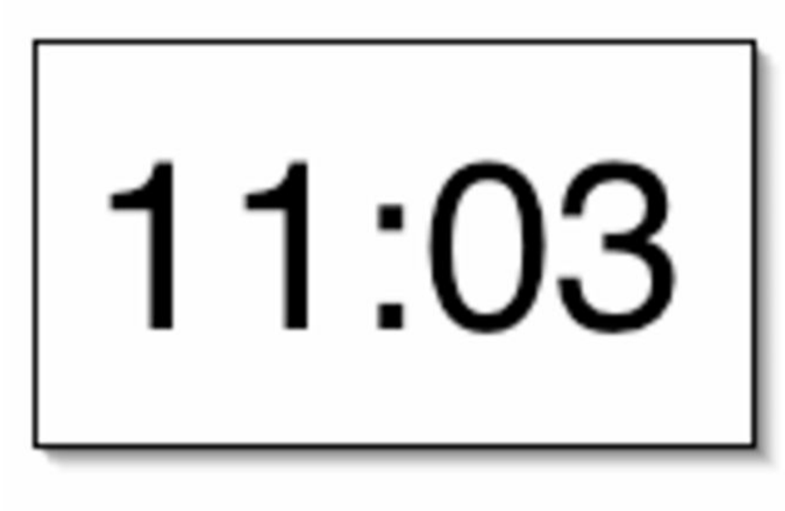
Abstraction and Modularization

Abstraction: the ability of ignoring details of the parts to focus at a higher level of the problem.

Modularization: the process of dividing the whole in well defined parts, that can be built and analyzed separately, and whose interaction is also given a well defined way.

A digital clock

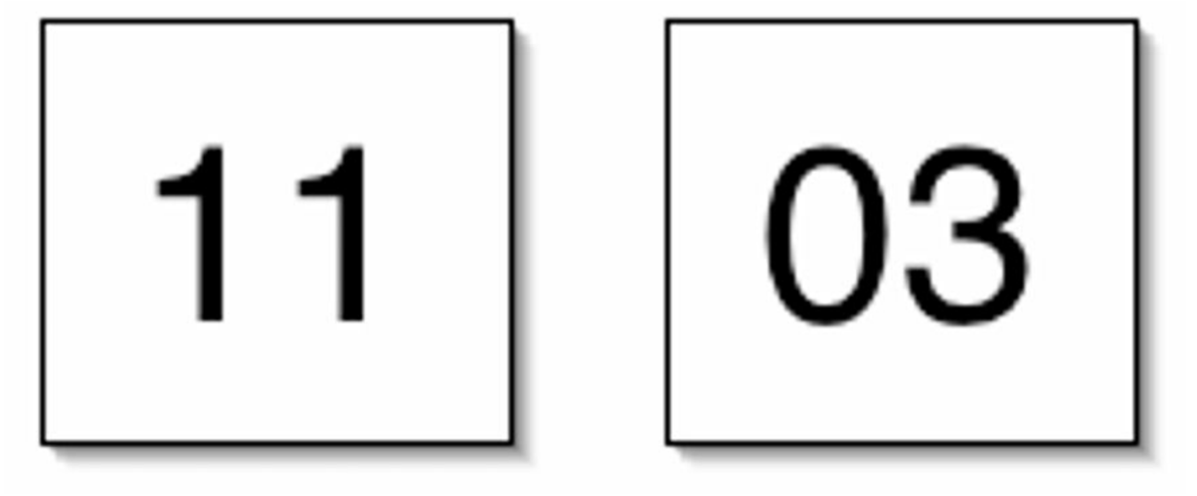
We want this



, but optionally we could also want this

A digital clock

We could just think about a two (or more) number displays



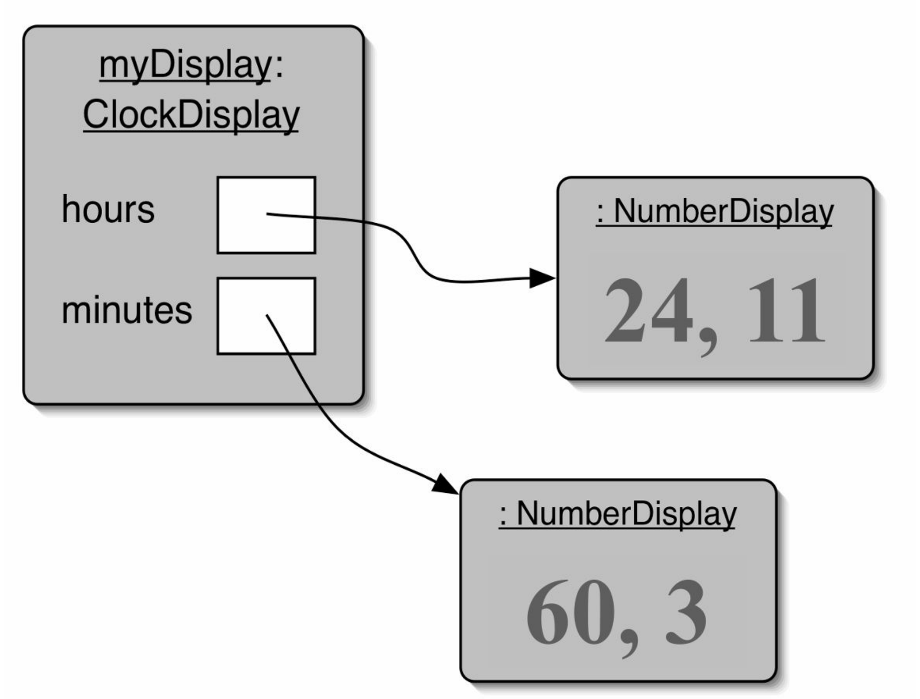
A digital clock

```
public class NumberDisplay {

  private int value;
  private int limit;

  //CONSTRUCTORS

  //METHODS

}
```

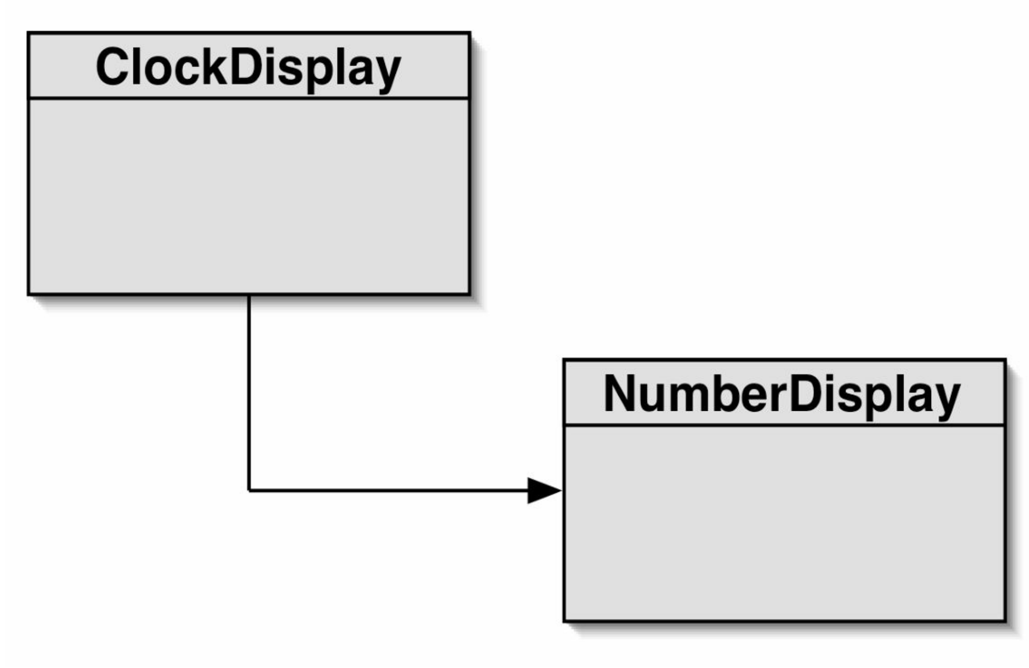
A digital clock

```
public class ClockDisplay {

  private NumberDisplay hours;
  private NumberDisplay minutes;

  //CONSTRUCTORS

  //METHODS

}
```

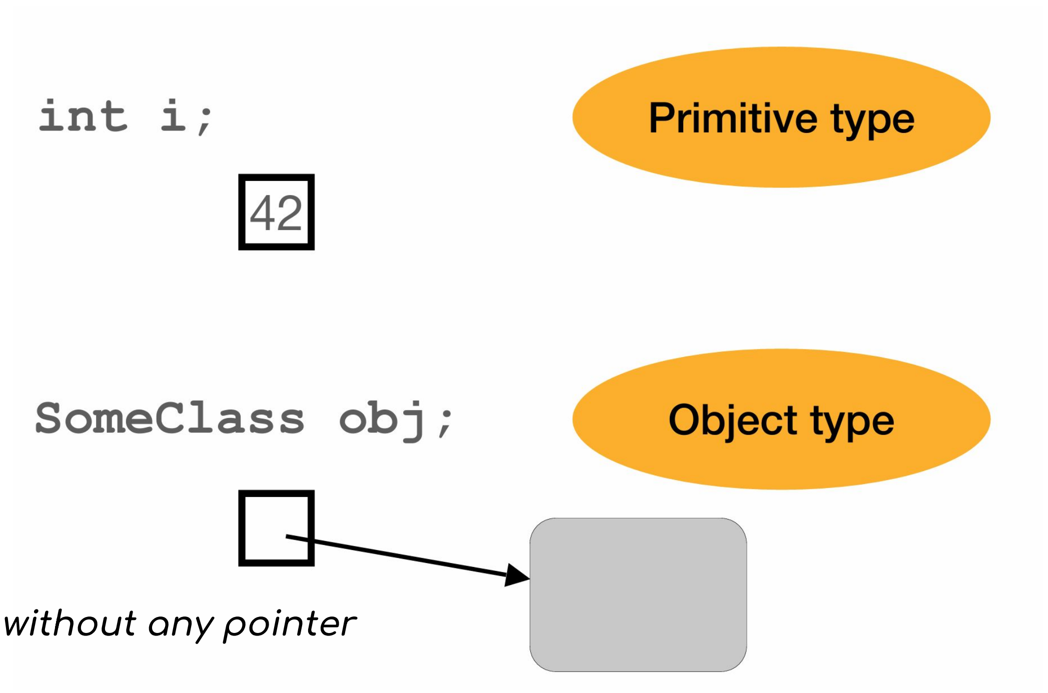
Object Diagram



Class Diagram



Primitive types vs Object types



It looks like C pointers without any pointer operators

Primitive types vs Object types - Example

What is the output for each case?

```
int a;
int b;
a = 42;
b = a;
a = a + 1;
System.out.println(a);
System.out.println(b);
```

```
Person a;
Person b;
a = new Person("Jules");
b = a;
a.changeName("Vincent");
System.out.println(a.getName());
System.out.println(b.getName());
```

Display Clock - Let’s work on it

Let’s code for a bit