Constructors

Example of definition

```
public class TimerClass {
    private int time; // in seconds

    public TimerClass() { // initializes time with the default value 0
    }
    public TimerClass(int minutes) {
        if (minutes < 0 || minutes > 60)
            throw new IllegalArgumentException();
        this.time = minutes * 60;
    }
    public TimerClass(TimerClass otherTimer) { // copy constructor
        this.time = otherTimer.time; // or otherTimer.getTime() ?
    }
...
}
```

Remark

In Java constructors can be overloaded

Constructors

Example of use

```
TimerClass t1 = new TimerClass();
TimerClass t2 = new TimerClass(42);
TimerClass t3 = new TimerClass(t2);
assert t1.getTime()==0 && t2.getTime()==42*60 &&
       t2.getTime()==t3.getTime();
```

Object creation and initialization in Java

Simplified rules

1. immediately after object creation a default value is assigned to each instance variable of the object
2. the default value is determined by the declared type of the instance variable:
   - 0 for **`int`** and other numerical types
   - **`false`** for **`boolean`**
   - **`null`** for reference types
3. if any, instance variable initializers for the class are executed in the left-to-right textual order
4. the constructor of the class matching the number and types of parameters is invoked

Object creation and initialization in Java

Example of variable initializer

```
public class TimerClass {
   private int time = 60; // in seconds, default is 1 minute

    public TimerClass() { // initializes time with the default value 60
    }
    public TimerClass(int minutes) {
        if (minutes < 0 || minutes > 60)
            throw new IllegalArgumentException();
        this.time = minutes * 60;
    }
    public TimerClass(TimerClass otherTimer) { // copy constructor
        this.time = otherTimer.time; // or otherTimer.getTime() ?
    }
...
}
```

Object creation and initialization in Java

Example

```
TimerClass timer1 = new TimerClass();
TimerClass timer2 = new TimerClass(1);
assert timer1.getTime() == timer2.getTime();
```

Object creation and initialization in Java

Overloaded constructors

- multiple constructors can be defined
- definitions must differ either in the number or in the type of the parameters

Default constructor

- implicitly defined if no constructor is provided (but only in that case)
- it has no parameters and the empty body

Explicit constructor invocation

- a constructor may be explicitly invoked in another constructor
- syntax: `'this' '(' (Exp ( ',' Exp)*)? ')'`
- rules: explicit invocation allowed only on the first line, cycles forbidden

Explicit constructor invocation

Example

```
public class Person {
    private String name; // not optional, cannot be null
    private String address; // optional, allowed to be null

    public Person(String name) {
        if (name == null) // instance variable name is not optional
            throw new NullPointerException();
        this.name = name;
    }
    public Person(String name, String address) {
        this(name); // calls the constructor with a single argument
        this.address = address;
    }
    public String getName() { // getter method
        return name;
    }
    public String getAddress() { // getter method
        return address;
    }
}
```

Explicit constructor invocation

Example

```
Person sam = new Person("Samuele");
Person sim = new Person("Simone","Genova");
assert sam.getAddress()==null && sim.getAddress()!=null;
```

Remarks

Non optional and optional fields

- in static OOL fields cannot be added or removed at runtime
- non optional fields must always have a defined value
- optional fields may have un undefined value
- in many OOL: undefined value=**`null`**

Strings in Java

- predefined class `String`
- strings are immutable objects
- string literals have a standard syntax

Class variables

Instance versus class variables

- instance variables = attributes of the objects
- **class variables** = attributes of the class
- instance variable in C = a different variable for each object of C
- class variable in C = a single variable for C

Java syntax and terminology

- Syntax:
  - field read: `CID '.'FID`
  - field update: `CID '.'FID '='Exp`

  `CID` class identifier, `FID` field identifier
- Terminology: *class variable*, or *static variable*, or *static field*

Class variables

Example

```
public class Item {
    private static long nextSN; // next unused serial number
    private int price; // in cents
    private long serialNumber;
    // invariant price >= 0 && serialNumber >= 0
    // for all o,o':Item if o.serialNumber == o'.serialNumber  then o==
        o'
    public Item(int price) {
        if (price < 0)
            throw new IllegalArgumentException();
        this.price = price;
        this.serialNumber = Item.nextSN++;
    }
    public int getPrice() {
        return this.price;
    }
    public long getSerialNumber() {
        return this.serialNumber;
    }
}
```

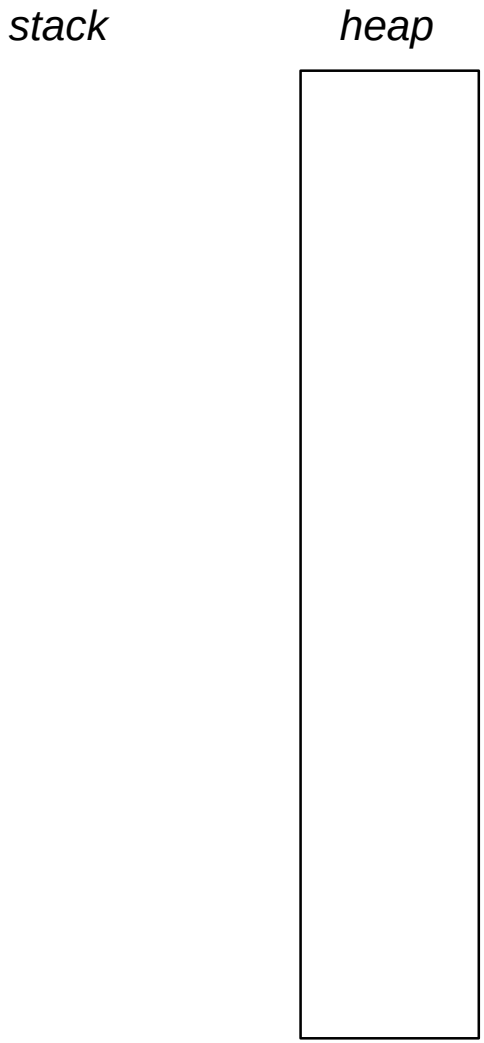
Class variables

Example

```
Item item1 = new Item(61_50);
Item item2 = new Item(14_00);
assert item1.getPrice() == 61_50 && item1.getSerialNumber() == 0;
assert item2.getPrice() == 14_00 && item2.getSerialNumber() == 1;
```

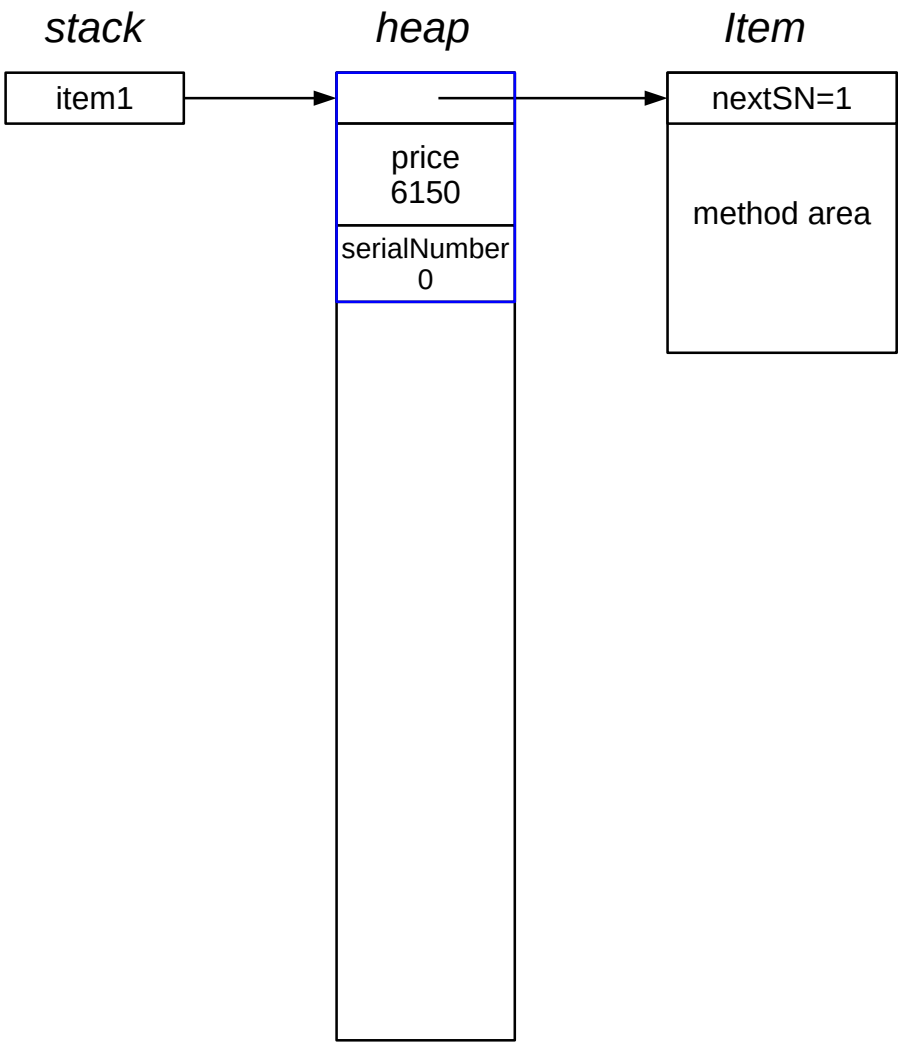
Example with memory model

```
Item item1 = new Item(61_50);
Item item2 = new Item(14_00);
```



Example with memory model

```
Item item1 = new Item(61_50); ⇐
Item item2 = new Item(14_00);
```



Example with memory model

```
Item item1 = new Item(61_50);
Item item2 = new Item(14_00); ⇐
```