Object values (1)

Objects as first-class values

In OOL objects are first-class values

- they can be assigned to variables
- passed as arguments to methods
- returned as values by methods

Object identity

Objects are represented by their identity
In practice:

identity = reference = address where the object is stored on the heap

Object values (2)

Assignment and argument passing

In most OOL (Java included) objects are assigned or passed by reference

Example

```
TimerClass t1 = new TimerClass();
TimerClass t2 = t1; // t2 and t1 refer to the same object
TimerClass t3 = null; // t3 refers to no object
assert t1 == t2 && t1 != t3;
```

Remarks

- `t2` contains the same object identity (that is, reference) as `t1`
- `==` tests whether two expressions evaluate to the same object identity
- **`null`**: useful constant value to denote “no object”
- `t3` refers to no object

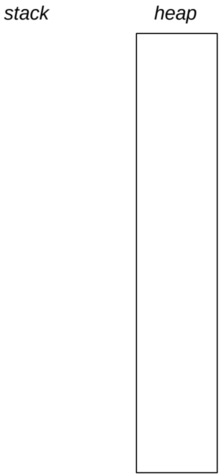
Classes

Another type of timer

```java
public class AnotherTimerClass {
   private int seconds;// invariant: 0<=seconds<=59
   private int minutes;// invariant: 0<=minutes<=60 && (minutes<60 || seconds==0)
   public boolean isRunning() {
      ...
   }
   public int getTime() {
      ...
   }
   public void tick() {
      ...
   }
   public int reset(int minutes) {
      ...
   }
}
```

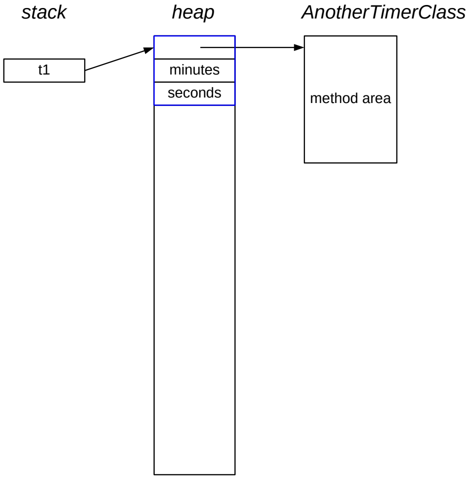
Example with memory model

```
                                   ⇐
AnotherTimerClass t1 = new AnotherTimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = new TimerClass();
TimerClass t4 = t3;
```



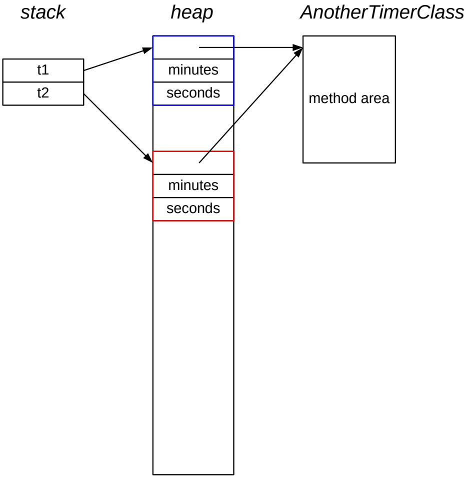
Example with memory model

```
AnotherTimerClass t1 = new AnotherTimerClass(); ⇐
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = new TimerClass();
TimerClass t4 = t3;
```



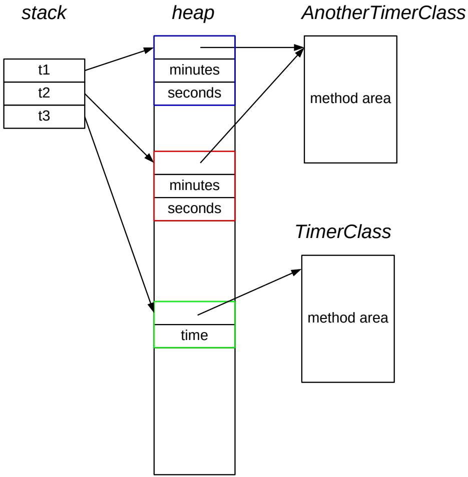
Example with memory model

```
AnotherTimerClass t1 = new AnotherTimerClass();
AnotherTimerClass t2 = new AnotherTimerClass(); ⇐
TimerClass t3 = new TimerClass();
TimerClass t4 = t3;
```



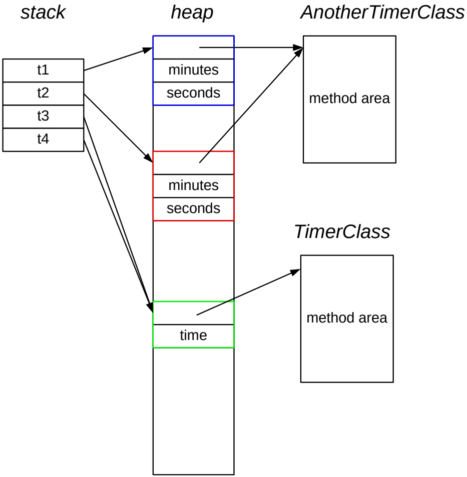
Example with memory model

```
AnotherTimerClass t1 = new AnotherTimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = new TimerClass(); ⇐
TimerClass t4 = t3;
```



Example with memory model

```
AnotherTimerClass t1 = new AnotherTimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = new TimerClass();
TimerClass t4 = t3;                ⇐
```



Object types and subtyping

In statically typed languages

- classes define also static types
- terminology: reference type (in Java), object type (in other contexts)
- example: `TimerClass`, `AnotherTimerClass`

Meaning of object types

If expression *e* has static type `TimerClass` then the evaluation of *e* will return

- either an instance of `TimerClass`
- or an instance of a subtype of `TimerClass`
- or the **`null`** reference

Remark

- inheritance and subtyping are two key concepts of OOP
- these notions will be introduced later on

Classes as static and dynamic types

Java checks types statically

Types `TimerClass` and `AnotherTimerClass` are not compatible

```
TimerClass t1 = new TimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = t2; // type error!
AnotherTimerClass t4 = t1; // type error!
```

In OOL types can be also checked dynamically

```
TimerClass t1 = new TimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = null;

assert t1 instanceof TimerClass && !(t1 instanceof AnotherTimerClass);
assert t2 instanceof AnotherTimerClass && !(t2 instanceof TimerClass);
assert !(t3 instanceof TimerClass)&&!(t3 instanceof AnotherTimerClass);
```

Design by contract

Code contracts: pre-conditions, post-conditions, invariants

- pre-condition for method *m*
  `requires` *p*: predicate *p* is required to hold immediately before the execution of *m*
- post-condition for method *m*
  `ensures` *p*: predicate *p* is required to hold immediately after the execution of *m* (if the pre-condition holds)
- invariant for class *C*: a predicate that is required to hold
  - immediately after creation of each instance of *C*;
  - immediately before the execution of each instance method of *C*;
  - immediately after the execution of each instance method of *C* (if the pre-condition holds).

Predicate definitions contain

- logical connectives and quantifiers
- the formal parameters and the result of a method, **`this`**, and the instance variables of the class

Design by contract

Example

```
public class TimerClass {
    private int time;
    /* invariant 0 <= time && time <= 3600; */

    public int reset(int minutes)
    /*  requires 0 <= minutes && minutes <= 60;
        ensures  result == old(this.time)
                 && this.time == minutes * 60; */
    {
        if (minutes < 0 || minutes > 60)
            throw new IllegalArgumentException();
        int prevTime = this.time;
        this.time = minutes * 60;
        return prevTime;
    }
    public boolean isRunning()
    /* ensures result == this.time > 0
               && this.time == old(this.time); */
    {
        return this.time > 0;
    }
    ...
}
```

How ensuring class invariants?

Data consistency in objects

- information hiding: the state of an object can be modified only through method access; no arbitrary changes are allowed!
- all methods must preserve invariants
- initially, the invariant must be verified by construction
- **constructors** are used for initializing object correctly, while guaranteeing information hiding