



Total Programming

Life cycle

Total programming



Assignment: Digital clocks

- ❑ Digital clocks register time in terms of hours and minutes
 - ❑ Methods to query the hours, respectively the minutes of a digital clock
 - ❑ A method to initialize a new digital clock with the lowest possible values for hours and minutes
 - ❑ Methods to register the hours, respectively the minutes of a digital clock to a given value
 - ❑ A method to initialize a new digital clock with given values for hours and minutes
 - ❑ A method to advance the time displayed by a digital clock with 1 minute
 - ❑ A method to check whether the time displayed by one digital clock is earlier than the time displayed by another digital clock
 - ❑ A method to synchronize the time displayed by a digital clock with the time displayed by another digital clock

Assignment: Digital clocks

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ In their current form, digital clocks display time on a 24-hour scale ranging between 0 and 23
 - ❑ The definition of the class must, however, be such that it is easy to add other formats for displaying time
 - An example of such a format is 12-hour display, using a range between 1 and 12, complemented with an AM/PM indicator

Assignment: Digital Clocks

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Construct a new digital clock displaying 7h35 as its time
 - ❑ A variable "myClock" will reference the new clock
- ❑ Construct another digital clock displaying 0h00 as its time
 - ❑ A variable "yourClock" will reference the new clock
- ❑ Change the hours of the clock referenced by the variable "myClock" to 33
- ❑ Introduce a variable "myMinutes" initialized with the minutes displayed by "myClock"
- ❑ Have the variable "yourClock" referencing the clock referenced by the variable "myClock"
- ❑ Advance the time displayed by the clock referenced by the variable "yourClock" with 1 minute

Class Definition: Steps



- ❑ **Specification**
 - ❑ Work out the information needed to manipulate objects of the class
 - Attach documentation comments to the class and to its methods
- ❑ **Representation**
 - ❑ Establish the information to be stored in each object of the class
 - Introduce instance variables and static variables
- ❑ **Implementation**
 - ❑ Work out an implementation for each of the methods of a class
 - Introduce additional methods to manage the complexity of the class
- ❑ **Verification**
 - ❑ Work out a consistent test suite to verify the correctness
 - This phase is not discussed in this session
- ❑ **Epilogue**

Specification: Guidelines



- ❑ **Work out proper documentation for each method of a class**
 - ❑ The documentation starts with a **heading** briefly explaining the intention of the method
 - The heading is used in an overview of all the methods of a class
 - ❑ The documentation is complemented with a more detailed description of the **effect** of the method
 - Use tags such as “@param” and “@return” to structure the documentation of methods
 - ❑ Documentation is worked out in a **natural language** (Dutch, ...)
 - In this course, we also use a **formal notation** based on first-order logic and set theory
- ❑ **Java offers “javadoc” as a tool to extract the documentation from the entire definition of a class**
 - ❑ “javadoc” generates documentation in the form of HTML-files
 - You can develop so-called “doclets” or “taglets” to learn “javadoc” how to process your own tags

Methods

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Distinguish between constructors, mutators and inspectors
 - ❑ Constructors initialize new objects
 - Destructors are introduced later as methods to destroy objects
 - ❑ Mutators change the state of objects; inspectors return information concerning the current state of objects
 - Methods at the same time changing the state of objects and returning information are bad practice (most of the time)
- ❑ Distinguish between instance methods and static methods
 - ❑ A static qualification informs the user of the class that the same effect is obtained for all objects of the class
 - Static methods are not easily turned into instance methods
- ❑ Use method overloading whenever different methods must have the same name
 - ❑ In Java, overloaded methods must differ in the number and/or type of their arguments

Basic Inspectors

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Basic inspectors of a class must reveal all the information concerning the current state of the class and its objects
 - ❑ Basic inspectors must be independent of each other
 - The set of basic inspectors may not include inspectors whose result can be specified in terms of other basic inspectors
- ❑ The semantics of all other methods of a class are specified in terms of its basic inspectors
 - ❑ Basic inspectors are not only used in formal specifications; informal specifications also appeal in an intuitive way to basic inspectors
 - In the end, all the specifications can be expanded such that only basic inspectors are used in them
- ❑ For more complex classes, several candidate sets of basic inspectors may exist
 - ❑ In choosing a proper set, non-total inspectors are best avoided as much as possible
 - The choice between several sets is often an arbitrary one, although it may have some impact on the specification of the class

Annotations: @Basic

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Since Java 1.5, ingredients of programs can be annotated
 - ❑ Annotations (also known as metadata) provide a formalized way to add information to code
 - Annotations can be available in source code, in class files or at runtime
 - You can develop annotation processors to process annotations
 - ❑ Annotations start with the symbol “@” followed by their name
 - Some annotations such as “@Override” and “@Deprecated” are predefined
 - Annotations are similar to modifiers such as “public” and “static”
 - ❑ Annotations must be defined in dedicated interfaces
 - Annotations may carry information (fields), complemented with methods for manipulating that information
- ❑ Basic inspectors are annotated “@Basic”
 - ❑ “@Basic” is a non-standard annotation

Annotations: @Immutable

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Inspectors without any arguments can be annotated “@Immutable”
 - ❑ Immutable inspectors always return the same result, even if their target object (or class) has changed its state
 - Clients of a class can safely work with a copy of the returned value

Task 1

Postconditions

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Postconditions describe the **effect** of methods offered by a class
 - ❑ Implementers of a class must see to it that all the postconditions are satisfied upon exit from each method
 - Postconditions express duties for the implementers of a class
 - ❑ Users of a class may assume that all the stated postconditions are satisfied upon return from a method invocation
 - Postconditions express rights for the users of a class
- ❑ Postconditions are worked in the **heading** of the method
 - ❑ Postconditions for mutators and constructors start with the tag “@post”
 - A “taglet” is needed to learn “javadoc” how to deal with the tag “@post”
 - ❑ Postconditions for non-basic inspectors start with the predefined tag “@return”
- ❑ The symbol “|” separates formal specifications from informal specifications

Postconditions: Details

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ In postconditions for mutators, the **state** of objects upon entry can be distinguished from their state upon exit
 - ❑ References to the **new state** of objects are qualified with the keyword “new”
 - Example: new.getBalance() == 10
 - Example: (new destination).getBalance() == 20
 - ❑ References to the **old state** of objects are unqualified
 - Example: getBalance() == 1000
 - Example: this.getBalance() == 1000
 - Example: destination.getBalance() == 200
- ❑ In postconditions for inspectors, the predefined identifier “**result**” is used to refer to the result they return
 - ❑ In postconditions for inspectors, there is no need to distinguish between old states and new states of objects
- ❑ In postconditions for constructors, references to the resulting state are also qualified “new”

Total Programming

KATHOLIEKE UNIVERSITEIT
LEUVEN

- A total method covers all possible exceptional cases in its effect
 - A total method always returns in a normal way to its caller
 - A total constructor always results in a properly initialized object
 - A total mutator always changes the state of zero or more objects
 - A total inspector always returns a proper result to its caller
 - Exceptional cases are caused by illegal values for arguments, by inappropriate states of objects involved, ...

Task 2

Complex Methods

KATHOLIEKE UNIVERSITEIT
LEUVEN

- The effect of mutators can be specified in terms of other mutators
 - Effect clauses state that the specified method has the same effect as the combined effects of the specified method invocations
 - In formal specifications of effect clauses, boolean operators are used to combine method invocations
 - Effect clauses start with the non-standard tag “@effect”
- The effect of constructors can be specified in terms of other constructors and mutators
 - Effect clauses attached to constructors may use the notations “this (...)” and “super(...)”
- The result of non-basic inspectors can be specified in terms of other inspectors

Annotations: @Model

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ By default, specifications of public methods can only use other public methods
 - ❑ Specifications of private methods may use both public methods and private methods
 - In general, the specification of a method may only use methods with the same access right or with a wider access right
- ❑ Private methods may simplify both informal and formal specifications of public methods
 - ❑ Private methods can be annotated “@Model”, indicating that they can also be used in specifications of public methods
 - By definition, private methods cannot be used in the implementation of methods outside their class

Task 3

15

Object Manipulation

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ In Java, variables of primitive types adopt value semantics
 - ❑ Variables of primitive types directly store a value of that type
 - Assignment to variables of primitive types register a copy of their right-hand side
 - Comparisons involving elements of primitive types use their values
- ❑ In Java, variables of class types adopt reference semantics
 - ❑ Variables of class types store a reference (a pointer) to an object of the class
 - Assignment to variables of class types take a copy of the pointer resulting from the evaluation of their right-hand side
 - Assignment to a variable of class type does not register a copy of the right-hand side object
 - Comparisons involving variables of reference types compare references
 - Comparisons involving variables of class type do not consider the state of objects

Task 4

Class Definition: Steps

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ **Specification**
 - ❑ Work out the information needed to manipulate objects of the class
 - Attach documentation comments to the class and to its methods
- ❑ **Representation**
 - ❑ Establish the information to be stored in each object of the class
 - Introduce instance variables and static variables
- ❑ **Implementation**
 - ❑ Work out an implementation for each of the methods of a class
 - Introduce additional methods to manage the complexity of the class
- ❑ **Verification**
 - ❑ Work out a consistent test suite to verify the correctness
 - This phase is not discussed in this session
- ❑ **Epilogue**

Variables

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ **Distinguish between instance variables and static variables**
 - ❑ Instance variables are **specific for each object** of the class
 - The memory assigned to each object of a class includes room to store all its instance variables
 - ❑ Static variables are **common to all the objects** of the class
 - The definition of a class implies the allocation of a common pool of memory in which its static variables are stored
- ❑ **Think about introducing some amount of redundancy in the stored information**
 - ❑ Redundancy improves run-time efficiency for inspectors; it turns mutators and constructors less efficient

Encapsulation

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Encapsulate all aspects related to the representation of objects
 - ❑ Encapsulation (or information hiding) leads to **better adaptable** software
 - By hiding details concerning the way information is stored, it is easy to turn to another representation for that information
- ❑ Introduce **setters** and **getters** for each stored characteristic
 - ❑ A **setter** is a mutator setting its characteristic to a given value
 - Setters must deal with invalid values supplied to them
 - ❑ A **getter** is an inspector returning the current value of its characteristic
 - ❑ For **immutable characteristics**, setters are left out
 - Immutable characteristics are initialized once and for all at the time an object is created
 - Variables representing immutable characteristics are qualified “final”

Initialization

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Upon entry to a constructor, non-final instance variables are initialized to the **default value** of their type
 - ❑ **Instance initialization blocks** and declarations of instance variables are ‘executed’ in the order in which they occur in the class
 - Instance initialization blocks and explicit initializations of instance variables are expanded in the body of constructors
 - The expansion only applies to constructors that do not invoke other constructors of their class
 - ❑ Instance variables can be further initialized in the body of a constructor
- ❑ **Final instance variables can only be assigned once in the entire initialization process**
 - ❑ Final instance variables can be assigned either in their declaration, in instance initialization blocks or in the body of constructors

Task 5

Class Definition: Steps

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ **Specification**
 - ❑ Work out the information needed to manipulate objects of the class
 - Attach documentation comments to the class and to its methods
- ❑ **Representation**
 - ❑ Establish the information to be stored in each object of the class
 - Introduce instance variables and static variables
- ❑ **Implementation**
 - ❑ Work out an implementation for each of the methods of a class
 - Introduce additional methods to manage the complexity of the class
- ❑ **Verification**
 - ❑ Work out a consistent test suite to verify the correctness
 - This phase is not discussed in this session
- ❑ **Epilogue**

Method Implementation

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ In the body of instance methods, the implicit object is accessible through the predefined identifier “this”
 - ❑ Unqualified references to instance variables and instance methods apply to the implicit object
 - “this” is a read-only variable initialized with a reference to the implicit object upon entry to the method
- ❑ Implement more complex methods in terms of more primitive methods
 - ❑ The implementation of all methods must be easy to understand (simplicity)
 - In the literature, 20 lines of code is considered an absolute maximum for the body of a method
 - ❑ Changes to more primitive methods automatically propagate towards more complex methods (adaptability)
 - The same aspect of a software system (e.g., the handling of a special case) must not be worked out at several places

Task 6

Class Definition: Steps

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ **Specification**
 - ❑ Work out the information needed to manipulate objects of the class
 - Attach documentation comments to the class and to its methods
- ❑ **Representation**
 - ❑ Establish the information to be stored in each object of the class
 - Introduce instance variables and static variables
- ❑ **Implementation**
 - ❑ Work out an implementation for each of the methods of a class
 - Introduce additional methods to manage the complexity of the class
- ❑ **Verification**
 - ❑ Work out a consistent test suite to verify the correctness
 - This phase is not discussed in this session
- ❑ **Epilogue**

Summary

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ **Specification**
 - ❑ **Postconditions** represent rights for the clients of a class, and duties for the implementers of a class
 - All aspects revealed in the documentation and/or in the signature of a method cannot be changed easily
- ❑ **Representation**
 - ❑ **Encapsulate** aspects of representation in getters and setters
 - Instance variables and static variables are always kept private
- ❑ **Implementation**
 - ❑ Define **more complex methods** in terms of more primitive methods
 - The strategy applies to specification as well as to implementation
- ❑ **Total Programming**
 - ❑ Deal with **exceptional cases** in postconditions of methods
 - Exceptional cases are turned into normal cases

Homework

KATHOLIEKE UNIVERSITEIT
LEUVEN

- ❑ Extend the class of digital clocks with a method to **synchronize 2 digital clocks**
 - ❑ Handle exceptional cases in the most simple way
- ❑ Extend the class of digital clocks such that time is **registered in terms of hours, minutes and seconds**
 - ❑ A constructor initializing a new digital clock with given hours, given minutes and given seconds
 - ❑ A method to query the seconds of a digital clock
 - ❑ A method to set the seconds of a digital clock to a given value
 - ❑ A method to advance the time displayed by a digital clock with 1 second
 - ❑ You may assume that seconds are always in the range 0..59 for all digital clocks