Zdroj 1: https://is.muni.cz/auth/el/1433/jaro2016/IA010/sample.pdf

1: 6 points

1 - algol 60

2 - PL/I

3 - Smalltalk

4 - IF and LOOP (a choice between two paths and logically controlled iterations)

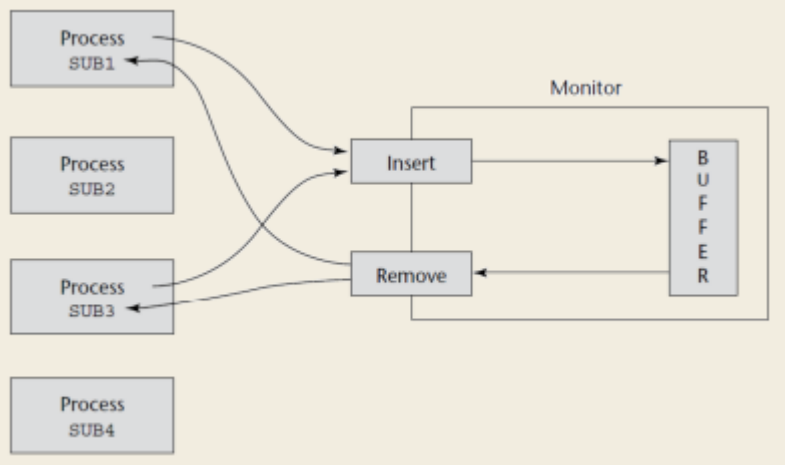
5 - once a keyword parameter appears, all following ones must be keyworded

6 - no code reuse

2: 4 points

http://stackoverflow.com/questions/7335950/semaphore-vs-monitors-whats-the-difference

- use encapsulation
    - shared data are packaged with operations on these data
    - the representation is hidden – basically a kind of ADT
    - access mechanism is part of the monitor
- if some subroutine of the monitor is running, all other calling tasks are put into the queue
- guarantee mutual exclusion
- can be implemented using semaphores (and vice versa)
- e.g. synchronized in Java



3: 10 points

1 – Unary: count++; Compound: a += b

2 – Coercion: implicit type conversion, for compatible types

float f; double d;

f = d; /* precision may be lost, undefined possible */

explicit type cast: Change of types, explicitly stated in the program code

n : integer; r : real;

r := real(n); /* requires run-time conversion */

n := integer(r); /* requires run-time conversion and check */

3 - class methods • can be used even if the class is not instantiated • operate directly on the class, not on its instances

class variables • common to all objects of a given class (single copy) • belong to class, not to an object

instance methods, instance variables

in Java/C++ class methods/variables are called static variables/methods (because of the static keyword)

4 - subtyping occurs when you derive an interface (method signatures/access points/ways of reacting to the outside world) from another whereas subclassing occurs when you derive an implementation (methods, attributes/internal state and inside logic) of a class from another class through inheritance

5 - • a task can be executed implicitly • the unit executing a task may not necessarily wait for it to finish • after a task is finished, the execution can continue from a different point than the point of task execution

4: 6 points

a) ‘a = ‘b list, ‘b = free type variable (’b must be of the “element of a list of type ’a” type)

b) does not have a finite solution

c) ‘a = bool, ‘b = bool, ‘c = bool -> bool

d) ‘a = ‘h -> ‘h -> ‘h -> int -> ‘h

‘b = ‘h -> ‘h

‘c = ‘h

‘d = int

‘e = ‘h -> int -> ‘h

‘f = ‘h

‘g = int -> ‘h

‘h = ‘h (? unknown)

5: 6 points

a) 0 10 11

b)

c)

d)

6: 6 points

a) static - In a 1000 99 In b 77 -3 In c 41 99, languages: C, C++, Java

dynamic - In a 1000 99 In b 1000 -3 In c 41 -3, languages: Snobol, APL, original Lisp

7: 10 points

Syntax:

```
try {
        // Code that might raise an exception
} catch (SomeException e) {
        // Handler body
} finally {
        // Finalizer body
}
```

• every catch must have a parameter

Finalization:

• the finally clause

- actions which should always be executed, e.g. • closing a file • releasing an external resource
- when executed:
  - if the exception is caught: after the handler completes
  - if the exception is not caught: before propagation
- in try block with no exception handlers: • in conjunction with break, continue and/or return

Classes of exceptions:

• class exception hierarchy (different from all other objects)

• all exceptions are objects of classes that are descendants of Throwable

- Error • errors thrown by the Java run-time system (e.g. running out of heap memory) • never thrown by user programs (and should not be handled there)
- Exception • by convention the parent class of user-defined exceptions
- Exception.RuntimeException • thrown by JVM, if a user program causes an error (e.g. ArrayIndexOutOfBoundsException or NullPointerException)
- Exception.IOException • I/O errors (methods in the java.io package)

Unchecked exceptions:

- classes Error and Exception.RuntimeException (including their subclasses)
- not a concern of the compiler reason: any method can throw them

Checked exceptions :

- all other exceptions

• all such exceptions a method can throw must be either caught in the method listed in method's header (throws)

• checking is done at compile time (different from C++)

Propagating checked exceptions:

• method which does not include a throws clause cannot propagate any exception

The propagation rules: If a method foo calls another method, which declares SomeException among its throws, then foo must either: catch and handle SomeException catch SomeException and throw some exception listed in the foo's throw clause list SomeException among its throws

• compiler-checked exception propagation

Zdroj 2:
https://www.facebook.com/groups/128695710522335/permalink/1113660328692530/?comment_id=1114114151980481

1:

1 – Binding – an association between a name and the object it names.

Name – a sequence of characters used to identify some object (entity)

Variable – an abstraction of a (collection of) computer memory cell

Object – anything which can have an associated name

2 - Coordinating access to shared resources

liveness • e.g. a ready task gets the processor in a finite amount of time • the computation proceeds towards a goal

deadlock • if two or more tasks wait for each other

3 - can partly substitute for multiple inheritance, one class can implement multiple interfaces

interfaces provide polymorphism: • can be treated as types • variables can be declared to be of an interface type • parameters can be declared to be of an interface type

solve the following problems: • methods of the same name in two parent classes • variable name conflicts (variables cannot be defined in interfaces)

disadvantages of interfaces: • no code reuse

new problems: • methods of the same name and protocol in multiple implemented interfaces

4 – Zdroj 1 otazka 3.3

2: https://is.muni.cz/auth/el/1433/jaro2016/IA010/um/01-history.pdf

3, 4, 5: Zdroj 1 otazky 4, 5, 6

6:

Mutex = competitive synchronisation: several tasks compete for a resource

methods for providing mutually exclusive access:

 semaphores: Zdroj 1 otazka 2

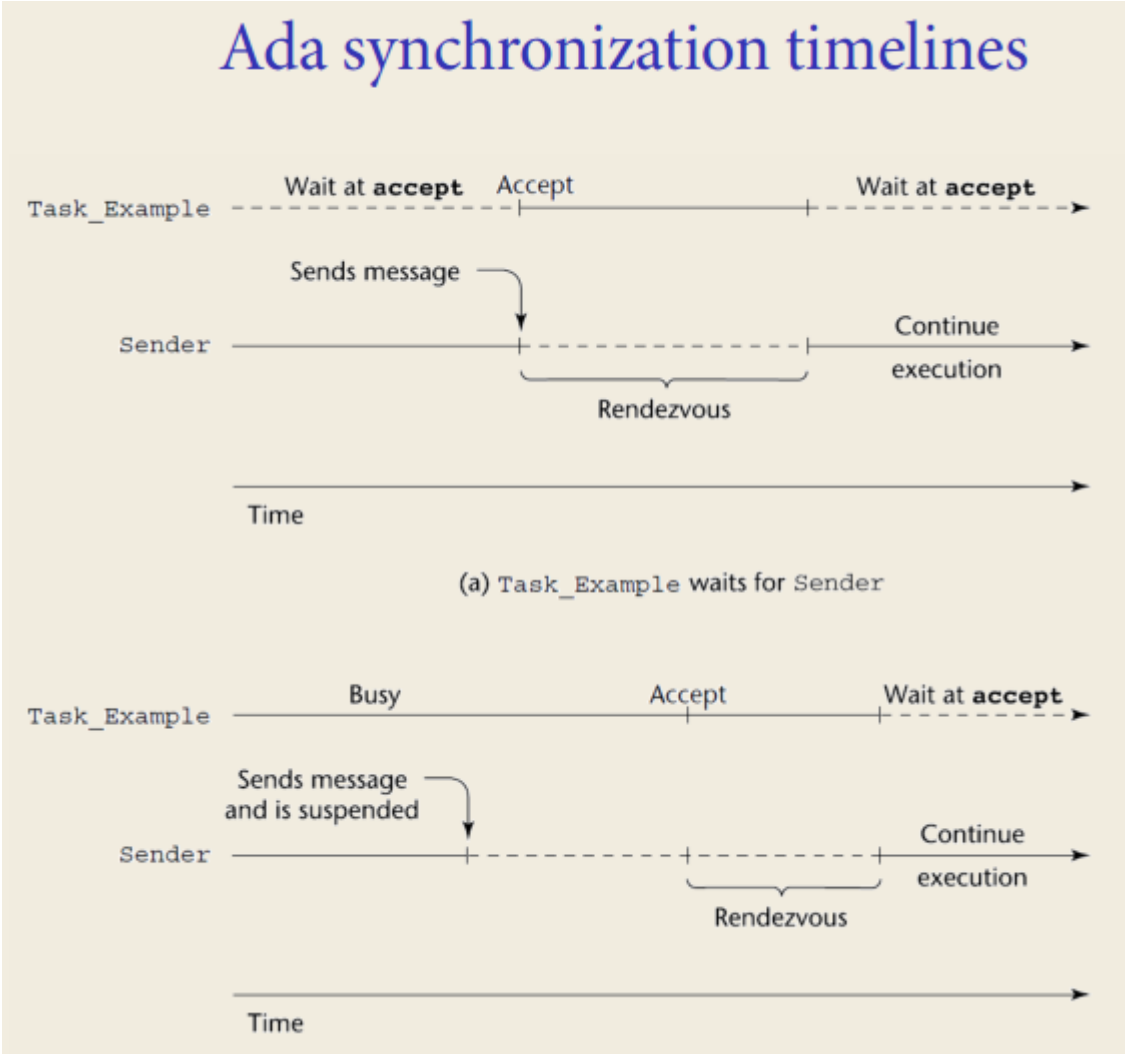
 monitors: Zdroj 1 otazka 2

 message passing:

• nondeterminism is needed to achieve fairness ⇒ related to guarded commands

• message passing can be synchronous or asynchronous

synchronous message passing • one of the tasks sends a message • the other task must be ready to receive the message • only then the communication takes place – rendezvous • the information exchange can then go in both directions



7:

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)#Static_and_dynamic_polymorphism