
EECE 276

Embedded Systems

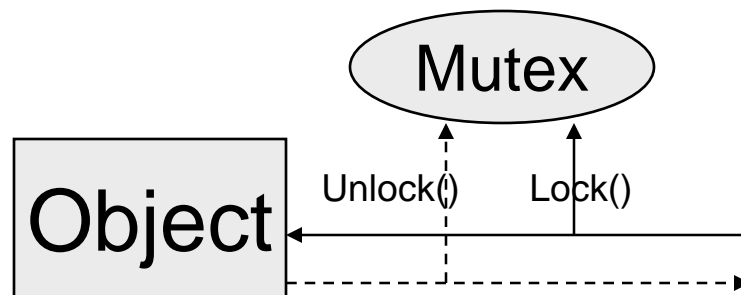
Issues in procedural and object-oriented languages

Object-Oriented Languages

- Primary “construction blocks”: objects
- Support:
 - » Data abstraction, inheritance, polymorphism, messaging
- Examples:
 - » C++, Java, C#, Eiffel, Ada 95

Synchronizing objects

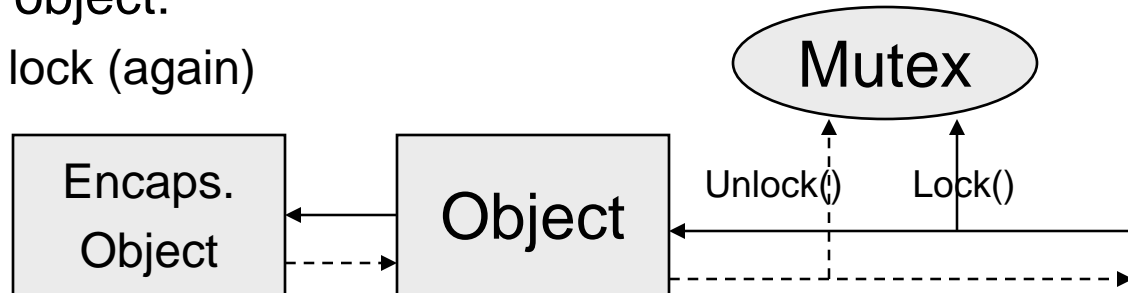
- Construction via object composition:
- Synchronization policies and object composition:
 - » Synchronized object:
 - Mutex associated with the object
 - Internal vs. external locking
 - Example: Java “synchronized” objects (internal locking)



Synchronizing objects

Synchronization policies and object composition:

- » Encapsulated object:
 - No need to lock (again)



- » Thread-local object
 - Accessed from a single thread – no locking needed
- » Object migration across threads
 - Objects don't have to be locked
 - Migration must be synchronized
- » Immutable objects (read-only)
- » Unsynchronized objects

Automatic memory management

- “Garbage”: allocated memory no longer used but not otherwise available
- Need for: garbage collection
 - » Converts “garbage” into reusable, “free”, memory
- Strategies
 - » Mark and sweep:
 - Mark reachable blocks, then sweep the rest (i.e.collect the garbage)
 - » Generational:
 - Objects are divided into groups based on their “age”
 - “Young” objects promoted to “old” age if they survive long enough
 - Members of the “old” generation are “condemned” less frequently

Automatic memory management

- GC Strategies
 - » Reference counting
 - Objects count how many references point to them (Cycles?)
 - Time-bounded!
 - » Copy:
 - Copy reachable objects into a new space, then swap spaces
 - » Mark-compact:
 - Mark reachable objects, then compact them into a block (the space used by the old objects becomes a contiguous free block)
 - » Stop-the-world
 - All threads are suspended while GC is running
- In general: GC performance is unpredictable!

Metrics and OO Languages

- Execution: less efficient than procedural
- Compilation: OK (about the same)
- Small-scale development: better!
- Large-scale development:
 - » OO has a different style – codified via Design Patterns
- Language features: worse (more complexity)

OO vs. Procedural Languages

- Main arguments against OO:
 - » Object composition and concurrency management
 - » Unpredictability of timing, performance
- Procedural languages:
 - » Everything is clearly defined, deterministic
- OO languages:
 - » “Dynamic binding”, complex object structures make properties hard to predict
 - » Inheritance anomaly: (inheritance + concurrency)
 - Synchronization code cannot be inherited w/o class rewriting.

OO vs. Procedural Languages

- Inheritance anomaly example:

```
class BoundedBuffer
    put: pre: not full
    get: pre: not empty
```

```
class MyBoundedBuffer : BoundedBuffer
    put: pre: not full
    get: pre: not empty and lastInvokedPut
```

- » MyBoundedBuffer has *violated* substitutability (Liskov)

- New directions:

- » Only limited concurrency patterns are used
- » Patterns are directly supported by an “platform” (RT-CORBA)
- » Systems are composed from components via *models*