Java: Just Another Version of Ada — an overview of Ada 2005 —

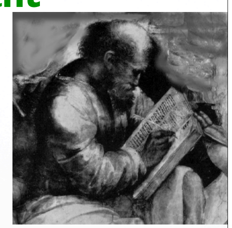
Jorge L. Díaz-Herrera, *Ph.D.*
Professor of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Tel: 585-475-4786 E-mail: jdiaz@gccis.rit.edu

Topics To Be Covered

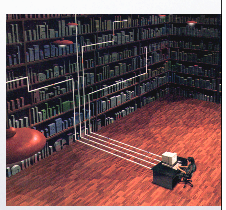
- ➡Introduction
  - Ada design goals
  - One-minute history
- Core Language
- Static Structure
- Dynamic Structure
- Ada vs. Real-time Java
- Conclusions

Ada Design Goals

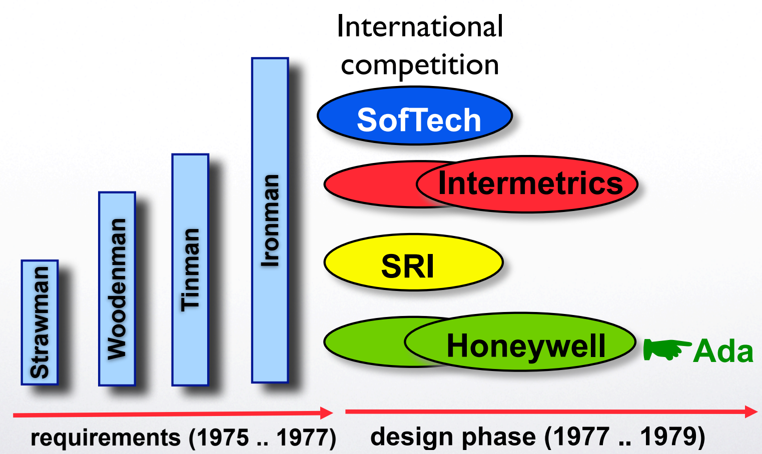
(1) Development as a human activity: **Want to write defect-free software?**
- A complex language, but simpler than C++ and even Java
- A self-contained deployment environment



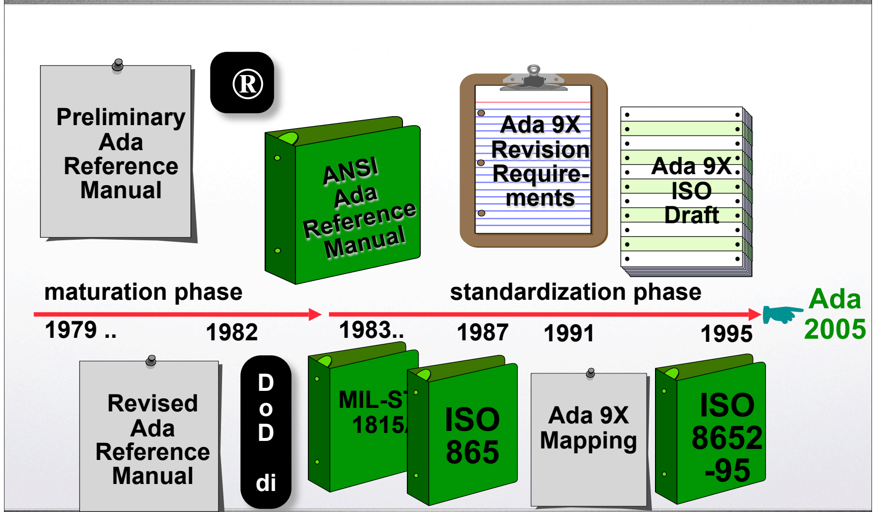
(2) From building to "growing" software
- Highly portable (does not depend on target platform)
- Component-based development
- Efficiency not an issue



A One-Minute History – I



A One-Minute History –2



The Ada Standard

Three documents

- Ada Reference Manual
- Annotated Ada Reference Manual
  - The language (13 chapters, ~550 pages)
  - The Standard Libraries
  - The 17 annexes (~500 pages)
- Ada Rationale
  - Programming paradigms
  - The core language
  - The annexes

Topics To Be Covered

✓Introduction

➡Core Language
- what is an Ada program?
- strong typing

Static Structure

Dynamic Structure

Ada vs. Real-time Java

Conclusions

What is an Ada "Program"?

"An Ada program is a set of *partitions*, each of which may execute in a separate address space, possibly on a separate computer...

a partition is constructed from *library units*."

- Composed of one or more *program units*
  - physically nested and hierarchically organized
- independently provided (program library) "Program text can be submitted in one or more compilations."

Core Language

Lexical elements: Block-structured, Strong typing, Exceptions, Typed signatures

Structural elements: Packages, Child units, Subprograms, and Interfaces

Object-oriented programming: Inheritance, encapsulation, dynamic binding, Identity, explicit overriding

Concurrent programming: Tasks, Synchronization, Priorities

Real-time and fault tolerance: Clocks, Scheduling control, Security, Hardware access

Distributed computing: Partitions (VN), RPC

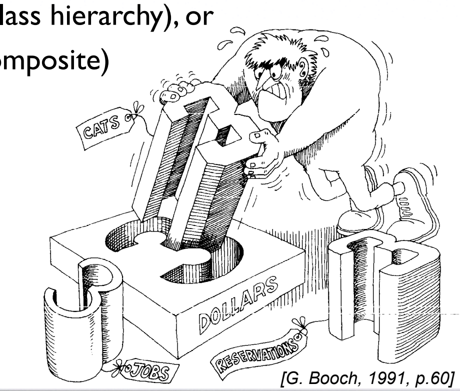
Types

In Ada, every type is either

- specific (a node in a class hierarchy),
- class-wide (an entire class hierarchy), or
- Universal (scalar or composite)

Strong Typing:
Each type in a "class" is identified by a tag, held by each object belonging to the type.

Access types



[G. Booch, 1991, p.60]

Topics To Be Covered

✓Introduction

✓Core Language

➡Static structure

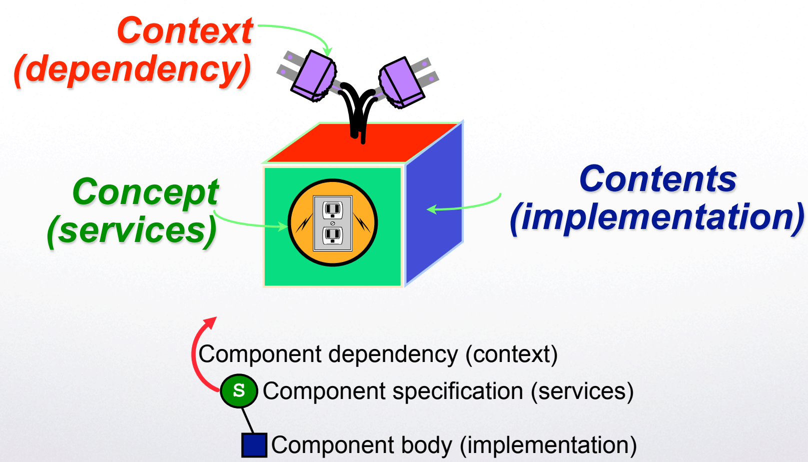
- design-by-composition
- design-by-decomposition
- design-by-extension
- design-by-adaptation

Dynamic structure

Ada vs. Real-time Java

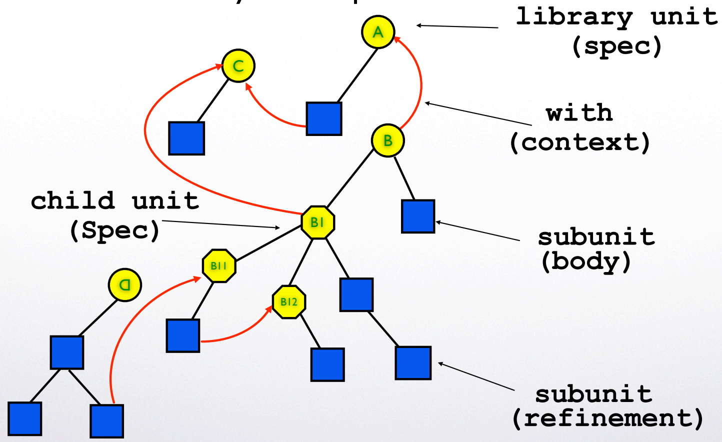
Conclusions

Software Components



Static Structure Summary

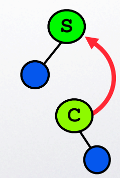
A Directed Acyclic Graph



Design-by-Composition

Component (C) becomes *client* of another *server* component (S) by *importing* services offered by the latter.

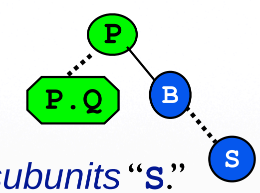
```
package S is
  -- exported (i.e., visible) specification
  end S;
  package body S is
  -- implementation follows
  end S;

with S;
  -- component C
  ...
  -- component C body
```



Design-by-Decomposition

Any library package "**P**" may have *child unit* "**P.Q**"

Any component body (i.e., *secondary unit*) "**B**" may contain local units implemented separately a.k.a. *subunits* "**S**."

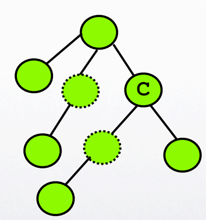


Results in a *tree-like parent-child hierarchy* of child units and/or subunits built top-down and rooted at root library unit "P"

Design-by-Extension

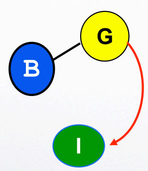
A component becomes an extension of another component by inheriting services offered by the latter.

Results in a *treelike class hierarchy* built strictly top down



Design-by-Adaptation

A component I is an instance of a “component template” G by providing parametric values specified by the latter.



- Instance specializes services
- Contract: If an actual parameter satisfies the requirements of the corresponding formal parameter, then a “body” B that matches the formal specification will work

Object-Oriented Programming

Preserve Ada’s strengths for building safe systems

- Distinction between specific and class-wide types
- Static binding by default, dynamic binding only when necessary
- Strong boundary around modules: A “class” is a package exporting a “tagged” type

Enhance object-oriented features

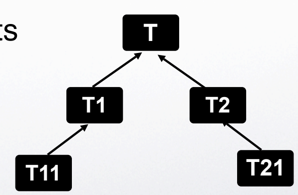
- Multi-package cyclic type structures
- Multiple-inheritance type hierarchies
- Concurrent OOP

Static Polymorphism

```
type T1 is new T; -- inherits from T
type T2 is new T; -- inherits from T
type T11 is new T1; -- inherits from T and T1
type T21 is new T2; -- inherits from T and T2
```

A ***class*** rooted at a type `T` consists of `T` and all of its derivatives

All types of a class rooted at `T` have at least the same set of operations as T

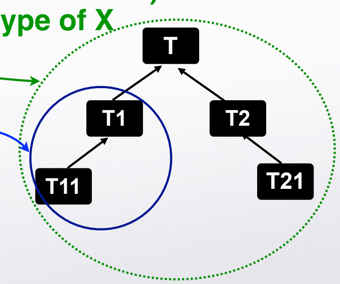


Dynamic Polymorphism

-- Class-wide operations

```
procedure Write (X: T'CLASS) is
begin ...
   Print (X) ;-- dispatches method at run-time
          -- (assuming Print is overloaded)
          -- based on "actual" type of X
end Write;
```

```
procedure Output
(X: T1'Class) is
begin
... (X) ...
```



Late (run-time) Binding

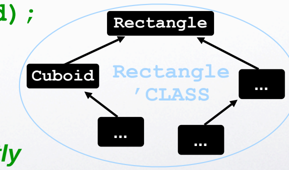
-- Access-To-Subprogram Types: subprograms as data

```
type Button is private;
type Resp is access procedure (B: T);
procedure Set_Up
         (B:out Button;R: Resp);
procedure Default (B: T);
...
type Button is record
    R : Resp := Default'ACCESS;
  ...
end record;
```


Generic "Class" Parameters

It is the combination of generics and inheritance that exploits the full potential of reuse

```
generic
 type T is new Rectangle;
 -- Rectangle operations imported implicitly
package Pk is ...;
package N_Pk is new Pk (Cuboid);
...

generic -- another example
  type B is new BOOLEAN;
 -- boolean operations imported implicitly
package Pk is ...;
```



Interfaces

Similar to abstract types but with multiple inheritance

- May be used as a secondary parent in type derivations
- Have class-wide types
- Support for composition of interfaces

Interfaces: Example

```
type Model is interface;

type Observer is interface;
procedure Notify (O: access Observer; M: access Model'Class)
        is abstract;

type View is interface and Observer;
procedure Display (V: access View; M: access Model'Class)
        is abstract;

type Controller is interface and Observer;
procedure Start (C: access Controller; M: access Model'Class)
        is abstract;

procedure Associate (V: accessView'Class;
        C: access Controller'Class; M: accessModel'Class);
```

Pascal Leroy, IBM

Interfaces: Example (cont'd)

```
type Device is tagged private;
procedure Input (D: in outDevice);

type Mouse is new Device
         and Controller with private;
procedure Input (D: in out Mouse);
procedure Start (D: access Mouse;
           M: access Model'Class);
procedure Notify (D: access Mouse;
           M : access Model'Class);
```


Concurrent OOP

Unify concurrent programming and object-oriented programming

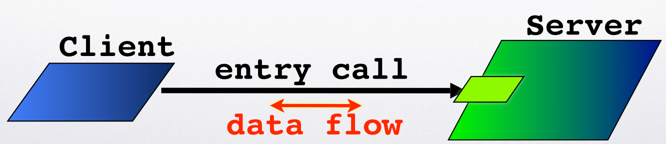
- Tasks are types (hence objects)
- Interfaces may specify synchronization properties
- Procedures may be implemented by task entries

```
task type Counter is
  entry Increase (By : POSITIVE);
  entry Decrease (By : POSITIVE);
  entry Get (Count : out NATURAL);
end Counter;
```


Task Interaction

Ada supports explicit task communication in the form of an essentially procedural interface between exactly two tasks

Achieved by a task (client) making entry calls to another task (server) accepting them



Topics To Be Covered

✓Introduction

✓Core Language

✓Static Structure

➡Dynamic Structure
- partitions and processes
- systems and real-time programming

Ada vs. Real-time Java

Conclusions

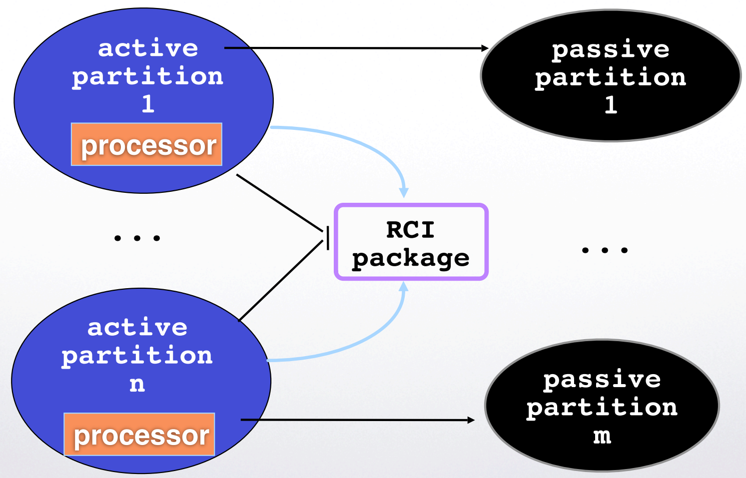
Distributed Ada Execution

An executable "system" is a cooperating set of

- one or more "active partitions"
- zero or more "passive partitions"

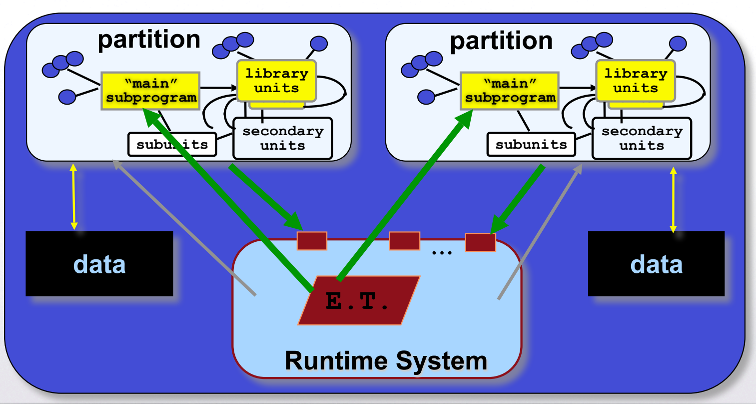
Partition: "a partition is a program or part of a program that can be invoked from outside the Ada implementation" 10.2/2

A Distributed Ada "System"



Heavyweight Process

The execution of an Ada "program" does not require an operating system



Topics To Be Covered

✓Introduction

✓Core Language

✓Static Structure

✓Dynamic Structure

➡Ada vs. Real-time Java

Conclusions

Ada & Real-time Programming

Language features promoting safety/reliability and deterministic language semantics (predictability)

Concurrency

- Well-defined semantics for scheduling
- Safe / efficient mutual exclusion, including "state notification"
- Safe / efficient coordination / communication

Hardware control

- Safe/predictable Memory management
- Asynchronous events / event handlers
- Asynchronous Transfer of Control (interrupts)
- Support for high-resolution time (millis and nanos), both absolute and relative
- Support for various kinds of timers, clocks
- Access to hardware-specific features

Java Summary

"Pure" Object-Oriented language in the style of Smalltalk

- Single inheritance of classes, "multiple inheritance" of "interfaces"

Built-in support for exception handling, threads

Well-defined semantics, at least for sequential features

- Classes are run-time objects
- All non-primitive data go on the heap

Emphasis on safety, security (downloadable "applets")

- Garbage collection required
- Portable, interpretable binary format for Java classes

"Core" libraries, and extensive set of "packages" for a wide variety of application domains

Java for Real-time? –1*

Thread method is error prone (Effect not always clear from source syntax)

- Requires cooperation by the accessing threads
  - Even if all methods are synchronized, an errant thread can access non-private fields without synchronization
  - A non-synchronized method may be safe to invoke from multiple threads, but a synchronized method might not be safe to invoke from multiple threads
  - Not always clear when a method needs to be declared as synchronized
- Complex interactions with other features (e.g. when are locks released?)
- Locking is hard to get right (exacerbated by absence of nested objects)

(*) Adapted from Ben Brosgol, Aonix

Java for Real-time? –2*

Limited mechanisms for direct inter-thread communication

- `wait()` and `notify()`/`notifyAll()` are low-level constructs that must be used very carefully
- Synchronized code that changes object's state must explicitly invoke `notify()`/`notifyAll()`
- No syntactic distinction between signatures of synchronized method that may suspend a caller and one that does not
- Only one wait set per object (versus per associated "condition")

Public thread interface issues

- The need to explicitly initiate a thread by invoking its `start()` method allows several kinds of programming errors
- Although `run()` is part of a thread class's public interface, invoking it explicitly is generally an error

(*) Adapted from Ben Brosgol, Aonix

Java for Real-time? –3*

Lack of some features useful for software engineering

- Operator overloading
- strongly typed primitive types, ...

Scheduling deficiencies

- Priority semantics are implementation dependent and fail to prevent unbounded *priority inversion*
- Section 17.12 of the Java Language Specification: "Every thread has a priority. ... threads with higher priority are ***generally*** executed in preference to threads with lower priority. Such preference is ***not, however, a guarantee that the highest priority thread will always be running***, and thread priorities cannot be used to reliably implement mutual exclusion."

Java for Real-time? –4*

Memory management unpredictability

- Predictable, efficient garbage collection appropriate for real-time applications not (yet) in mainstream
- lacks stack-based objects
- Heap used for exceptions thrown implicitly as an effect of other operations

Asynchrony deficiencies

- Event handling requires dedicated thread
- interrupt()not sufficient
- stop()and destroy()deprecated or dangerous or both

Java for Real-time? –5*

OOP has not been embraced by the real-time community

- Dynamic binding complicates analyzability
- Garbage Collection defeats predictability
- A class's "interface" is more than its public and protected members

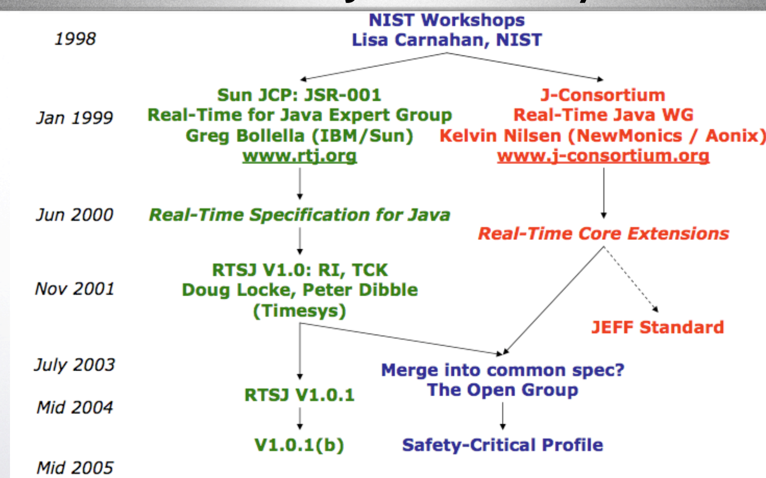
No features for accessing underlying hardware

Performance questions

"Standard" API would need to be rewritten for predictability

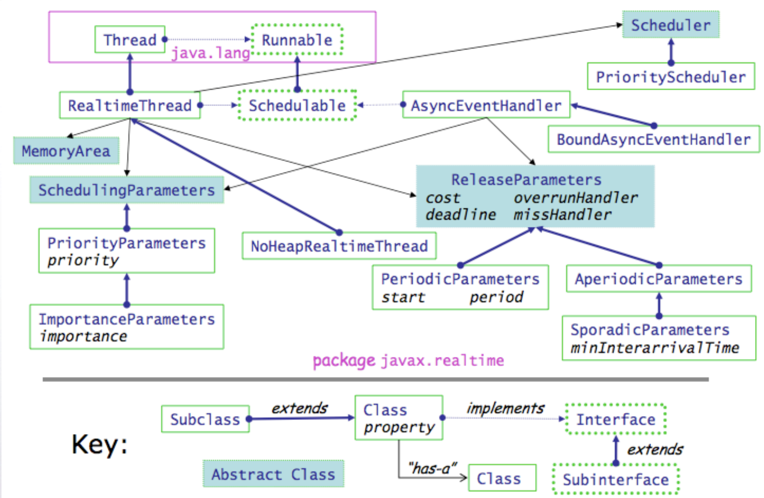
- In general it includes some implementation characteristics E.g. does it allocate objects, can it block
- Some JVM opcodes require non-constant amount of time

Real-time Java History*



Focus here will be on the Real-Time Specification for Java

Scheduling-related classes (partial list)*



Topics To Be Covered

✓Introduction

✓Core Language

✓Static Structure

✓Dynamic Structure

✓Ada vs. Real-time Java

➡Conclusions

Conclusions

Ada

- easier to "restrict" for building safety-critical systems (the features that makes creating solid applications possible)
- very successful in the safety-critical domain (high reliability military and space applications)

Java

- many safety-critical issue are intrinsic (pure OOP)
- C-based syntax prone to errors (hybrid type system)
- has not be used in the safety-critical domain

In Summary

Ada is a much better technical solution for implementing safety-critical distributed, concurrent systems

- powerful, semantically complete, well-designed
- There are a number of compilers including commercial development systems (AdaCore, Aonix, Artisan Software, Green Hills Software, IBM, and Polyspace technologies)

There are some deficiencies

- Availability of Ada programmers

Ada is worth another look!

The Future: Ada 2005 and beyond

The JTC1/SC22/WG9 ISO Working Group in charge of maintaining the Ada Language
http://www.open-std.org/JTC1/SC22/WG9/

AdaRapporteur Group collecting Ada Issues
http://www.ada-auth.org/arg-minutes.html

Ada Conformity Assessment Authority
http://www.ada-auth.org/

Resources

GNAT Academic Program (Open source)
http://www.adacore.com/home/academia/
http://libre2.adacore.com

SIGAda WWW Server Home Page
http://www.acm.org/sigada/

Ada Home: The Web Site for Ada
http://www.adahome.com/

Ada CORBA Products
http://www.adapower.com/corba/

A#: Ada for .NET
http://www.usafa.af.mil/df/dfcs/bios/mcc_html/a_sharp.cfm

Resources–2

Aonix
http://www.aonix.com

Artisan Software
http;//www.artisansw.com

Green Hills Software
http://www.ghs.com

IBM
http://www.ibm.com

Polyspace Technologies
http://www.polyspace.com

Comparison Chart*

Programming Structure, Modularity	Ada 83	Ada 95	Ada 2005
Packages	√	√	√
Child units		√	√
Limited with clauses and mutually dependent specs			√
Generic units	√	√	√
Formal packages		√	√
Partial parametrization			√

Object-Oriented Programming	Ada 83	Ada 95	Ada 2005
Derived types	√	√	√
Tagged types		√	√
Multiple inheritance of interfaces			√
Named access types	√	√	√
Access parameters, Access to subprograms		√	√
Enhanced anonymous access types			√
Aggregates	√	√	√
Extension aggregates		√	√
Aggregates of limited type			√
Unchecked deallocation	√	√	√
Controlled types, Accessibility rules		√	√
Accessibility rules for anonymous types			√

(*) from Adacore technologies

Comparison Chart*

Concurrency	Ada 83	Ada 95	Ada 2005
Tasks	√	√	√
Protected types, Distributed annex		√	√
Synchronized interfaces			√
Delays, Timed calls	√	√	√
Real-time annex		√	√
Ravenscar profile, Scheduling policies			√

Scientific Computing	Ada 83	Ada 95	Ada 2005
Numeric types	√	√	√
Complex types		√	√
Vector/matrix libraries			√

Standard Libraries	Ada 83	Ada 95	Ada 2005
Input/output	√	√	√
Elementary functions		√	√
Containers			√

Character Support	Ada 83	Ada 95	Ada 2005
7-bit ASCII	√	√	√
8/16 bit		√	√
8/16/32 bit (full unicode)			√

(*) from Adacore technologies