Ada SYNTAX CARD

bold	Ada keyword	*italic*	Ada 95
[ ]	Optional term	{ }	Repeatable
|	Alternative	\ \	Choose one
...	Identical term	::=	Expansion term

LIBRARY

← COMPILATION_UNIT ::=
  {**with** library_unit_name {,...}**;** | use_clause} library_item
| {**with** library_unit_name {,...}**;** | use_clause} **separate (**parent_name**)**
    \subprogram_body|package_body|task_body|protected_body\

← USE_CLAUSE ::=
  use pack_name {,...}**;** | **use type** subtype_name {,...}**;**

← LIBRARY_ITEM ::=
  [***private***] subprogram_spec**;** | [***private***] package_spec**;**
| [***private***] **generic** {generic_formals|use_clause} subprogram_spec**;**
| [***private***] **generic** {generic_formals|use_clause} package_spec**;**
| [***private***] **package** [*parent_name*.]id **is new** gen_pack_name
    [generic_actuals]**;**
| [***private***] **procedure** [*parent_name*.]id **is new** gen_proc_name
    [generic_actuals]**;**
| [***private***] **function** [*parent_name*.]id|op **is new** gen_func_name
    [generic_actuals]**;**
| subprogram_body**;** | package_body**;**
| [***private***] ***package*** [*parent_name*.]id ***renames*** pack_name**;**
| [***private***] ***generic package*** [*parent_name*.]id ***renames***
    gen_pack_name**;**
| [***private***] ***generic procedure*** [*parent_name*.]id ***renames***
    gen_proc_name**;**
| [***private***] ***generic function*** [*parent_name*.]id ***renames***
    gen_func_name**;**
| [***private***] subprogram_spec ***renames*** callable_entity_name**;**

DECLARATIONS

← BASIC_DECLARATION ::=
  type id **is (** \id|'character'\ {**,**...}**);**
| ***type*** id ***is mod*** static_expr**;**
| **type** id **is digits** static_expr [**range**static_s_expr **..** static_s_expr]**;**
| **type** id **is** [**delta** static_expr] **range** static_s_expr **..** static_s_expr**;**
| **type** id **is delta** static_expr ***digits*** static_expr [**range**
    static_s_expr **..** static_s_expr]**;**
| **type** id [discrim] **is** [***abstract***] **new** subtype_id [***with record*** list
    end record]**;**
| **type** id [discrim] **is** [***abstract***] **new** subtype_id [***with null record***]**;**
| **type** id **is** array_type_defn**;**
| **type** id [discrim] **is** [[***abstract***] ***tagged***] [***limited***] **record** list
    end record;
| ***type*** id [discrim] ***is*** [***abstract***] ***tagged*** [***limited***] ***null record***;
| **type** id **is access** [***all*** | ***constant***] subtype_id**;**
| ***type*** id ***is access*** [***protected***] ***procedure*** [formals]**;**
| ***type*** id ***is access*** [***protected***] ***function*** [formals] ***return***
    subtype_name**;**
| **task type** id [discrim] [**is**
  {**entry** id [**(**discrete_range**)**] [formals]**;** | rep_clause}
  [***private*** {**entry** id [**(**discrete_range**)**] [formals]**;** | rep_clause}]
  end [id]]**;**
| ***protected type*** id [discrim] ***is***
  { subprogram_spec | ***entry*** id [**(**discrete_range**)**] [formals]**;** | rep_clause }
  [***private*** { subprogram_spec | ***entry*** id [**(**discrete_range**)**] [formals]**;** |
      id {**,**id} **:** [***aliased***] subtype_id [**:=** expr]**;** | rep_clause }]
  end [id]**;**
| **type** id [**(<>)**|discrim]**;**
| **type** id [**(<>)**|discrim] **is** [[***abstract***] ***tagged***] [**limited**] **private;**
| ***type*** id [**(<>)**|discrim] ***is*** [***abstract***] ***new*** ancestor_subtype_id ***with private***;
| **subtype** id **is** subtype_id**;**
| id {**,**id} **:** [***aliased***] [**constant**] subtype_id [**:=** expr]**;**
| id {**,**id} **:** [***aliased***] [**constant**] array_type_defn [**:=** expr]**;**
| **task** id [**is**
  {**entry** id [**(**discrete_range**)**] [formals]**;** | rep_clause}
  [***private*** {**entry** id [**(**discrete_range**)**] [formals]**;** | rep_clause}]
  end [id]]**;**
| ***protected*** id ***is***
  { subprogram_spec | ***entry*** id [**(**discrete_range**)**] [formals]**;** | rep_clause }
  [***private*** { subprogram_spec | ***entry*** id [**(**discrete_range**)**] [formals]**;** |
      id {**,**id} **:** [***aliased***] subtype_id [**:=** expr]**;** | rep_clause }]
  end [id]**;**
| id {**,**id} **: constant :=** static_expr**;**
| subprogram_spec [***is abstract***]**;**
| package_spec**;**
| id : subtype_name **renames** object_name**;**
| id : exception **renames** exception_name**;**
| **package** id **renames** pack_name**;**
| subprogram_spec **renames** callable_entity_name**;**
| ***generic package*** id ***renames*** gen_pack_name**;**
| ***generic procedure*** id ***renames*** gen_proc_name**;**
| ***generic function*** id ***renames*** gen_func_name**;**
| id {**,**id} **: exception;**
| **generic** {generic_formals|use_clause} subprogram_spec**;**
| **generic** {generic_formals|use_clause} package_spec**;**
| **package** id **is new** gen_pack_name [generic_actuals]**;**
| **procedure** id **is new** gen_proc_name [generic_actuals]**;**
| **function** id|op **is new** gen_func_name [generic_actuals]**;**

← SUBTYPE_ID ::=
  subtype_name
| subtype_name **range** name'Range[**(**static_expr**)**]
| subtype_name **range** s_expr **..** s_expr
| subtype_name [**digits**|**delta**] static_expr [**range** name'Range[**(**static_expr**)**]
| subtype_name [**digits**|**delta**] static_expr [**range** s_expr **..** s_expr]
| subtype_name **(**discrete_range {**,**...}**)**
| subtype_name **(**[selector_name {**|**...} **=>**] expr {**,**...}**)**

← ARRAY_TYPE_DEFN ::=
  array(subtype_name **range <>** {**,**...}**)** of [***aliased***] subtype_id
| **array(**discrete_range {**,**...}**)** of [***aliased***] subtype_id

← DISCRETE_RANGE ::=
  discrete_subtype_id | name'Range[**(**static_expr**)**] | s_expr **..** s_expr

← DISCRIM ::=
  (id {**,**id} : [***access***] subtype_name [**:=** expr] {**;** ...}**)**

← LIST ::=
  id {**,**id} **:** [***aliased***] subtype_id [**:=** expr]**;** | rep_clause {...}
| {id {**,**id} **:** [***aliased***] subtype_id [**:=** expr]**;** | rep_clause {...}}
  case name **is**
   when \expr|discrete_range|**others**\ {| ...} **=>** list
   {...}
  end case;
| **null;**

← DECLARATIVE_ITEM ::=
  basic_declarative_item
| subprogram_body | package_body | task_body | protected_body
| subprogram_spec **is separate;** | **package body** id **is separate;**
| **task body** id **is separate;** | ***protected body*** id ***is separate***;

← BASIC_DECLARATIVE_ITEM ::=
  basic_declaration | rep_clause | use_clause

← SUBPROGRAM_SPEC ::=
  procedure [*parent_name*.]id [formals]
| **function** [*parent_name*.]id|op [formals] **return** subtype_name

← FORMALS ::=
  (id {**,**id} **:** [**in** | **in out** | **out** | ***access***] subtype_name [**:=** expr] {**;** ...}**)**

← SUBPROGRAM_BODY ::=
  subprogram_spec **is**
   {declarative_item}
  begin handled_statements
  end [id]**;**

← PACKAGE_SPEC ::=
  package [*parent_name*.]id **is**
   {basic_declarative_item}
  [**private** {basic_declarative_item}]
  end [[*parent_name*.]id];

← PACKAGE_BODY ::=
  package body [*parent_name*.]id **is**
   {declarative_item}
  [**begin** handled_statements]
  end [[*parent_name*.]id]**;**

← TASK_BODY ::=
  task body id **is**
   {declarative_item}
  begin
   handled_statements
  end [id]**;**

← PROTECTED_BODY ::=
  protected body id ***is***
   { subprogram_spec | subprogram_body |
     entry id1 [**(*for*** id2 ***in*** discrete_range**)**] [formals] ***when*** bool_expr ***is***
      {declarative_item}
     begin handled_statements
     end [id1]**; |**

```
   rep_clause }
 end [id];

← GENERIC_FORMALS ::=
  id {,id} : [in] subtype_name [:= expr];
| type id[(<>)|discrim] is [[abstract] tagged] [limited] private;
| type id[(<>)|discrim] is [abstract] new subtype_name [with private];
| type id is (<>);
| type id is range <>;
| type id is mod <>;
| type id is digits <>;
| type id is delta <> [digits <>];
| type id is array_type_defn;
| type id is access [all | constant] subtype_id;
| type id is access [protected] procedure [formals];
| type id is access [protected] function [formals]
            return subtype_name;
| with subprogram_spec [is \name|<>\];
| with package id is new gen_pack_name \(<>)|[generic_actuals]\;

← GENERIC_ACTUALS ::=
  ([selector_name =>] \expr|var_name|subprog_name|entry_name|
                      subtype_name|pack_inst_name\ {,...})
```

STATEMENTS, EXPRESSIONS

```
← NAME ::=
  id          | op                  | name.all
| name(expr {,...})                 | name(discrete_range)
| name.selector_name                | name'attribute_designator
| subtype_name(expr|name)           | 'character'
| func_name [([selector_name =>] \expr|var_name\ {,...})]

← SELECTOR_NAME ::=
  id | 'character' | op

← ATTRIBUTE_DESIGNATOR ::=
  id[(static_expr)] | Access | Delta | Digits

← AGGREGATE ::=
  array_aggregate
| ([\expr|subtype_name\ with] [selector_name {|...} =>
            | others => ] expr {,...})
| ([\expr|subtype_name\ with] null record)

← ARRAY_AGGREGATE ::=
  (expr, expr {,...})               | (expr {,...}, others => expr)
| (\expr|discrete_range|others\ {|...} => expr {,...})

← EXPR ::=
  relation {xor relation}
| relation {and relation}           | relation {and then relation}
| relation {or relation}            | relation {or else relation}

← RELATION ::=
  s_expr [\= | /= | < | <= | > | >=\  s_expr]
| s_expr [not] in name'Range[(static_expr)]
| s_expr [not] in s_expr .. s_expr
| s_expr [not] in subtype_name

← S_EXPR ::=
  [+|-] term {\+|-|&\ term}

← TERM ::=
  factor {\*|/|mod|rem\ factor}

← FACTOR ::=
  primary [** primary] | abs primary | not primary

← PRIMARY ::=
  numeric_literal | null | string_literal | aggregate | name
| subtype_name'(expr) | subtype_name'aggregate | new subtype_id
| new subtype_name'(expr) | new subtype_name'aggregate | (expr)

← STATEMENT ::=
  [<<label>>] program_statement

← PROGRAM_STATEMENT ::=
| var_name := expr;                | exit [loop_name] [when bool_expr];
| goto label;                      | null;
| return [expr];                   | entry_call_statement
| delay_statement                  | requeue entry_name [with abort];
| abort task_name {,...};          | raise [exception_name];
| subtype_name'(expr);             | subtype_name'aggregate;
| proc_name [([selector_name =>] \expr|var_name\ {,...})];
| if bool_expr then
    statement {...}
   {elsif bool_expr then statement {...}}
   [else statement {...}]
   end if;
| case expr is
    when \expr|discrete_range|others\ {| ...} => statement {...}
    {...}
   end case;
| [id:] [while bool_expr | for id in [reverse] discrete_range] loop
    statement {...}
   end loop [id];
| [id:] [declare {declarative_item}]
   begin handled_statements
   end [id];
| accept id [(expr)] [formals] [do handled_statements end [id]];
| select_statement

← HANDLED_STATEMENTS ::=
    statement {...}
   [exception
    when [id:] \exception_name|others\ {| ...} => statement {...}
    {...}]

← ENTRY_CALL_STATEMENT ::=
  entry_name [([selector_name =>] \expr|var_name\ {,...})];

← DELAY_STATEMENT ::=
  delay [until] delay_expr;

← SELECT_STATEMENT ::=
  select
   [when bool_expr =>]
     accept id [(expr)] [formals] [do handled_statements end [id]];
     [statement {...}]
   | delay_statement [statement {...}]
   | terminate;
  { or
   [when bool_expr =>]
     accept id [(expr)] [formals] [do handled_statements end [id]];
     [statement  {...}]
   | delay_statement [statement {...}]
   | terminate; }
  [ else statement {...} ]
  end select;
| select
    entry_call_statement [statement {...}]
   or delay_statement [statement {...}]
  end select;
| select
    entry_call_statement [statement {...}]
  else statement {...}
  end select;
| select
    \entry_call_statement|delay_statement\ [statement {...}]
  then abort  statement {...}
  end select;
```

REPRESENTATION

```
← REP_CLAUSE ::=
  for local_name'attribute_designator use expr;
| for local_name'attribute_designator use name;
| for first_subtype_local_name use array_aggregate;
| for first_subtype_local_name use record [at mod static_expr;]
   {component_name at static_expr range static_s_expr..static_s_expr;}
  end record;
| for \id|op\ use at expr;

← LOCAL_NAME ::=
  id['attribute_designator] | op['attribute_designator] | library_unit_name
```

LEXICAL

```
id                ::= identifier_letter {[underline] letter_or_digit}
letter_or_digit   ::= identifier_letter | digit
numeric_literal   ::= numeral [.numeral] [exponent] | numeral
                      #base [.base] # [exponent]
numeral           ::= digit {[underline] digit}
exponent          ::= E [[+]|-] numeral
base              ::= extended_digit {[underline] extended_digit}
extended_digit    ::= digit | A | B | C | D | E | F
string_literal    ::= "{"" | non_quote_character}"
comment           ::= --
op                ::= "<" | ">" | "=" | "&" | "<=" | ">=" | "+" | "/" | "*"
pragma            ::= pragma id [([id =>] \name|expr\ {,...})];
```