



Genericity

university.adacore.com

The notion of a pattern

- An algorithm can be abstracted from some types & subprograms

```
procedure Swap_Int (Left, Right : in out Integer) is
  V : Integer;
begin
  V := Left;
  Left := Right;
  Right := V;
end Swap_Int;
```

```
procedure Swap_Bool (Left, Right : in out Boolean) is
  V : Boolean;
begin
  V := Left;
  Left := Right;
  Right := V;
end Swap_Bool;
```

- It would be nice to extract these properties in some common pattern, and then just replace the parts that need to be replaced

```
procedure Swap (Left, Right : in out (Integer | Boolean)) is
  V : (Integer | Boolean);
begin
  V := Left;
  Left := Right;
  Right := V;
end Swap;
```

Solution: generics

- A generic unit is a unit that doesn't exist
- It is a pattern based on properties
- The instantiation applies the pattern to certain parameters

```
generic
  type T is private;
  procedure Swap (L, R : in out T)

  procedure Swap (L, R : in out T)
  is
    Tmp : T := L
  begin
    L := R;
    R := Tmp;
  end Swap;

  procedure Swap_I is new Swap (Integer);
  procedure Swap_F is new Swap (Float);

  I1, I2 : Integer;
  F1, F2 : Float;

  procedure Main is
  begin
    Swap_I (I1, I2);
    Swap_F (F1, F2);
  end Main;
```

```
template <class T>
void Swap (T & L, T & R);

template <class T>
void Swap (T & L, T & R) {
  T Tmp = L;
  L = R;
  R = Tmp;
}

int I1, I2;
float F1, F2;

void Main (void) {
  Swap <int> (I1, I2);
  Swap <float> (F1, F2);
}
```

What can be made generic?

- Subprograms & packages can be made generic
- Children of generic units have to be generic themselves

```
generic
  type T is private;
package Parent is [...]

generic
package Parent.Child is [...]

package I is new Parent (Integer);
package I_Child is new I.Child;
```

- Generic instantiation is creating new set of data

```
generic
  type T is private;
package P is
  V : T;
end P;

package I1 is new P (Integer);
package I2 is new P (Integer);

begin

  I1.V := 5;
  I2.V := 6;

  if I1.V /= I2.V then
    -- will go there
```

Generic types parameters


- A generic parameter is a template
- It specifies the properties the generic body can rely on

```
generic
  type T1 is private; -- this should have the properties of a private type
                        -- (assignment, comparison, ability to declare variables on the stack...)
  type T2 (<>) is private; -- this type can be unconstrained
package Parent is [...]
```

- The actual parameter must provide at least as many properties as the generic contract
- The usage in the generic has to follow the contract

```
generic
  type T (<>) is private;
procedure P (V : T);

procedure P (V : T)
is
  X1 : T := V; -- OK, we can constrain the object by initialization
  X2 : T;      -- Compilation error, there is no constraint for this object
begin [...]
```

 **Compilation error:** The object X2 is declared as type T but is not constrained by initialization, which is not allowed for a generic parameter.

```
procedure P1 is new P (String); -- OK, unconstrained objects are accepted
procedure P2 is new P (Integer); -- OK, the object is already constrained
```

Properties that can be expressed on generic types

- **private** – any definite (and non-limited) type
- **(<> private** – allowed to be indefinite
- **(<>)** – any discrete (integer or enumeration)
- **range <>** – any integer
- **digits <>** – any float
- **array** – array type (needs index and components)
- **access** – access type (needs target)

```
generic
  type T is (<>);
  function Add_One (V : T) return T is
  begin
    return T'Succ (V);
  end Add_One;

  procedure Add_One_I is new Add_One (Integer);
  procedure Add_One_C is new Add_One (Character);
```

Generic parameters can be built one on top of the other

- **Consistency is checked at compile-time**

```
generic
  type T is private;
  type Index is (<>);
  type Arr is array (Index range <>) of T;
procedure P;

type Int_Array is array (Character range <>) of Integer;

procedure P_String is new P
  (T      => Integer,
   Index => Character,
   Arr    => Int_Array);
```

Generic constants & variables parameters

- **Variables can be specified on the generic contract**
- **The mode specifies the way the variable can be used:**
 - in → read only
 - in out → read write
- **Generic variables can be defined after generic types**

```
generic
  type T is private;
  X1 : Integer;
  X2 : in out T;
  procedure P;

V : Float;

procedure P_I is new P
  (T => Float,
   X1 => 42,
   X2 => V);
```


Generic subprograms parameters

- Subprograms can be defined in the generic contract
- Must be introduced by “with” to differ from the generic unit

```
generic
  with procedure Callback;
procedure P;

procedure P is
begin
  Callback;
end P;

procedure Something;

procedure P_I is new P (Something);
```

- “is <>” – matching subprogram is taken by default
- “is null” – null subprogram is taken by default

```
generic
  with procedure Callback_1 is <>;
  with procedure Callback_2 is null;
procedure P;

procedure Callback_1;

procedure P_I is new P; -- Will take Callback_1 and null
```



? Quiz



Is this correct?

(1/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G; use G;

procedure P is
  package I is new G (Integer);
begin
  V := 0;
end P;
```



Is this correct?

(1/10)



NO

```
generic
  type T is private;
package G is
  V : T;
end G;
```



```
with G; use G;

procedure P is
  package I is new G (Integer);
begin
  V := 0;
end P;
```

The use clause cannot be made on a generic package (there's no actual instance)
On top of that, V is not directly visible because I is not used.



Is this correct?

(2/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G;

procedure P is
  type My_Integer is new Integer;

  package I1 is new G (Integer);
  package I2 is new G (My_Integer);

  use I1, I2;
begin
  V := 0;
end P;
```



Is this correct?

(2/10)



NO

```
generic
  type T is private;
package G is
  V : T;
end G;
```

Compilation error.

There is an ambiguity between the two
V (from I1 and from I2).

Prefix or qualification could work, e.g.:

I2.V := 0;

or

V := My_Integer'(0);

```
with G;

procedure P is
  type My_Integer is new Integer;

  package I1 is new G (Integer);
  package I2 is new G (My_Integer);

  use I1, I2;
begin
  V := 0;
end P;
```



V := 0;



Is this correct?

(3/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G;

procedure P is
  type My_Integer is new Integer;

  package I1 is new G (Integer);
  package I2 is new G (My_Integer);

  use I1;
begin
  V := 0;
end P;
```



Is this correct?

(3/10)



YES

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G;

procedure P is
  type My_Integer is new Integer;

  package I1 is new G (Integer);
  package I2 is new G (My_Integer);

  use I1;
begin
  V := 0;
end P;
```

Everything is OK here, I1.V will be assigned 0



Is this correct?

(4/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  V : in out Integer;
package P is
  V2 : Integer := V;
end P;
```

```
with P;

procedure Main is
  package I1 is new P (10);
  V1 : Integer := 20;
begin
  V2 := V1;
end Main;
```



Is this correct?

(4/10)



NO

```
generic
  V : in out Integer;
package P is
  V2 : Integer := V;
end P;
```

```
with P;
```



```
procedure Main is
  package I1 is new P (10);
  V1 : Integer := 20;
begin
  V2 := V1;
end Main;
```

The specification of V is "in out", it expects a variable



Is this correct?

(5/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
package G is

end G;

generic
package G.Child is
  V : T;
end G.Child;
```

```
with G;

procedure P is
  package I1 is new G (Integer);
begin
  I1.Child.V := 0;
end P;
```



Is this correct?

(5/10)



NO

```
generic
  type T is private;
package G is

end G;

generic
package G.Child is
  V : T;
end G.Child;
```

```
with G;

procedure P is
  package I1 is new G (Integer);
begin
  I1.Child.V := 0;
end P;
```



Compilation error.

Child needs to be instantiated separately, e.g.:

```
package Child1 is new I1.Child;
Child1.V := 0;
```



Is this correct?

(6/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T (<>) is private;
package G is
  V : T;
end G;
```

```
with G;

procedure P is
  package I1 is new G (Integer);
begin
  I1.V := 0;
end P;
```



Is this correct?

(6/10)



NO



```
generic
  type T (<>) is private;
package G is
  V : T;
end G;
```

```
with G;

procedure P is
  package I1 is new G (Integer);
begin
  I1.V := 0;
end P;
```

Compilation error.

T is known as indefinite (<>). So it's not possible to declare a variable V without specifying the constrain.



Is this correct?

(7/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G;

package P is
  type My_Type is private;

  package I1 is new G (My_Type);
private
  type My_Type is null record;
end P;
```



Is this correct?

(7/10)



NO

```
generic
  type T is private;
package G is
  V : T;
end G;
```

```
with G;

package P is
  type My_Type is private;
  package I1 is new G (My_Type);
private
  type My_Type is null record;
end P;
```



Compilation error.

A package is instantiated at the point of declaration.
In this case, we don't have the implementation of
My_Type yet, so we can't instantiate the package.



Is this correct?

(8/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T is private;
  procedure P;

  type R is record
    null;
  end record;

  type A is access all R;

  procedure I1 is new P (Integer);
  procedure I2 is new P (Float);
  procedure I3 is new P (Character);
  procedure I4 is new P (String);
  procedure I5 is new P (R);
  procedure I6 is new P (A);
```



Is this correct?

(8/10)



NO

```
generic
  type T is private;
  procedure P;

  type R is record
    null;
  end record;

  type A is access all R;

  procedure I1 is new P (Integer);
  procedure I2 is new P (Float);
  procedure I3 is new P (Character);
  procedure I4 is new P (String);
  procedure I5 is new P (R);
  procedure I6 is new P (A);
```



Compilation error.

I4 doesn't compile, because String is indefinite, and T expects a definite type.



Is this correct?

(9/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
  type T (<>) is private;
  procedure P;

  type R is record
    null;
  end record;

  type A is access all R;

  procedure I1 is new P (Integer);
  procedure I2 is new P (Float);
  procedure I3 is new P (Character);
  procedure I4 is new P (String);
  procedure I5 is new P (R);
  procedure I6 is new P (A);
```



Is this correct?

(9/10)



YES

```
generic
  type T (<>) is private;
procedure P;

type R is record
  null;
end record;

type A is access all R;

procedure I1 is new P (Integer);
procedure I2 is new P (Float);
procedure I3 is new P (Character);
procedure I4 is new P (String);
procedure I5 is new P (R);
procedure I6 is new P (A);
```

OK.

Here T can accept both definite and indefinite types.



Is this correct?

(10/10)



YES

(click on the check icon)

NO

(click on the error location(s))

```
generic
package P is
  type T is range 0 .. 10;
end P;
```

```
with P;

procedure Main is
  package I1 is new P;
  package I2 is new P;

  V1 : I1.T := 0;
  V2 : I2.T;
begin
  V2 := V1;
end Main;
```



Is this correct?

(10/10)



NO

```
generic
package P is
  type T is range 0 .. 10;
end P;
```

```
with P;

procedure Main is
  package I1 is new P;
  package I2 is new P;

  V1 : I1.T := 0;
  V2 : I2.T;
begin
  V2 := V1;
end Main;
```



Error I1.T and I2.T are two different types.



university.adacore.com