```
with Map_Site;
with Room;
with Maze;
with Create_Maze;
with Maze_Factory;
procedure Basic_Maze is

    The_Factory : Maze_Factory.Object;
    My_Basic_Maze : Maze.Pointer := Create_Maze (The_Factory);
    The_Room : Room.Object'Class :=
       Maze.Room_Of (1, In_The_Maze => My_Basic_Maze.all).all;

    use Map_Site;

begin
    Room.Enter (The_Room);

    for The_Direction in Room.Direction loop
        declare
            My_Maze_Location : Map_Site.Object'Class
               renames Room.Get_Side (The_Room, The_Direction).all;
        begin
            Enter (My_Maze_Location);
        end;
    end loop;
end Basic_Maze;
```

```
with Map_Site;
with Room;
with Maze;
with Create_Maze;
with Maze_Factory.Bombed;
procedure Bombed_Maze is

    The_Factory : Maze_Factory.Bombed.Object;
    My_Basic_Maze : Maze.Pointer := Create_Maze (The_Factory);
    The_Room : Room.Object'Class :=
       Maze.Room_Of (1, In_The_Maze => My_Basic_Maze.all).all;

    use Map_Site;

begin
    Room.Enter (The_Room);

    for The_Direction in Room.Direction loop
        declare
            My_Maze_Location : Map_Site.Object'Class
               renames Room.Get_Side (The_Room, The_Direction).all;
        begin
            Enter (My_Maze_Location);
        end;
    end loop;
end Bombed_Maze;
```

```
with Maze;
with Maze_Factory;
function Create_Maze
            (Using_Factory : in Maze_Factory.Object'Class) return Maze.Pointer;
```

```
with Map_Site;
with Room;
with Wall;
with Door;
with Maze;
with Maze_Factory;
function Create_Maze (Using_Factory : in Maze_Factory.Object'Class)
                     return Maze.Pointer is
    Result : Maze.Pointer := Maze_Factory.Make_Maze (Using_Factory);
    Room1 : Room.Pointer := Maze_Factory.Make_Room (Using_Factory, 1);
    Room2 : Room.Pointer := Maze_Factory.Make_Room (Using_Factory, 2);
    A_Door : Door.Pointer := Maze_Factory.Make_Door
                                (Using_Factory, Room1, Room2);

begin
    Maze.Add (Room1, Result.all);
    Maze.Add (Room2, Result.all);

    Room.Set_Side (Room1.all, Room.North,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));
    Room.Set_Side (Room1.all, Room.East, Map_Site.Pointer (A_Door));
    Room.Set_Side (Room1.all, Room.South,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));
    Room.Set_Side (Room1.all, Room.West,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));

    Room.Set_Side (Room2.all, Room.North,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));
    Room.Set_Side (Room2.all, Room.East,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));
    Room.Set_Side (Room2.all, Room.South,
                   Map_Site.Pointer (Maze_Factory.Make_Wall (Using_Factory)));
    Room.Set_Side (Room2.all, Room.West, Map_Site.Pointer (A_Door));

    return Result;
end Create_Maze;
```

```
with Map_Site;
with Room;
package Door is
    type Object is new Map_Site.Object with private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Door : in out Object);
    procedure Create (The_Door : in out Object; Room1, Room2 : in Room.Pointer);

    function Other_Side (Of_Door : in Object;
                         From_The_Room_Of : in Room.Pointer)
                        return Room.Pointer;

private
    type Object is new Map_Site.Object with
        record
            Is_Open : Boolean := False;
            Side1, Side2 : Room.Pointer;
        end record;
end Door;
```

```
with Map_Site;
with Ada.Text_Io;
package body Door is
    procedure Put_Line (Item : in String) renames Ada.Text_Io.Put_Line;

    procedure Enter (The_Door : in out Object) is
    begin
        Put_Line ("Entered Door between room number: " &
                  Natural'Image (Room.Number_Of (The_Door.Side1.all)) &
                  " and " &
                  Natural'Image (Room.Number_Of (The_Door.Side2.all)));
    end Enter;
    procedure Create (The_Door : in out Object;
                      Room1, Room2 : in Room.Pointer) is
    begin
        The_Door := (Map_Site.Object with
                     Is_Open => False,
                     Side1 => Room1,
                     Side2 => Room2);
    end Create;
    function Other_Side (Of_Door : in Object;
                         From_The_Room_Of : in Room.Pointer)
                        return Room.Pointer is
    begin
        if Room.Number_Of (Of_Door.Side1.all) =
           Room.Number_Of (From_The_Room_Of.all) then
            return Of_Door.Side2;
        elsif Room.Number_Of (Of_Door.Side2.all) =
              Room.Number_Of (From_The_Room_Of.all) then
            return Of_Door.Side2;
        else
            Put_Line
               ("Error: Received room that does not match either side of door");
            return null;
        end if;
    end Other_Side;
end Door;
```

```
package Map_Site is
    type Object is abstract tagged private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Object : in out Object) is abstract;

private
    type Object is abstract tagged null record;
end Map_Site;
```

```
with Room;
package Maze is
    type Object is tagged private;
    type Pointer is access all Object'Class;

    procedure Add (The_Room : in Room.Pointer;
                   To_The_Maze : in out Object'Class);

    function Room_Of (Number : in Positive; In_The_Maze : in Object'Class)
                     return Room.Pointer;

private
    type List_Of_Room_Type is array (1 .. 2) of Room.Pointer;
    type Object is tagged
        record
            List_Of_Rooms : List_Of_Room_Type;
        end record;
end Maze;
```

```
with Room;
with Ada.Text_Io;
package body Maze is
    procedure Put_Line (Item : in String) renames Ada.Text_Io.Put_Line;

    procedure Add (The_Room : in Room.Pointer;
                   To_The_Maze : in out Object'Class) is
        Room_Number : constant Positive := Room.Number_Of (The_Room.all);
    begin
        if Room_Number in To_The_Maze.List_Of_Rooms'Range then
            To_The_Maze.List_Of_Rooms (Room_Number) := The_Room;
        else
            Put_Line ("Error: Received room that is not part of the maze. " &
                      "Its number is: " & Natural'Image (Room_Number));
        end if;
    end Add;

    function Room_Of (Number : in Positive; In_The_Maze : in Object'Class)
                     return Room.Pointer is
    begin
        if Number in In_The_Maze.List_Of_Rooms'Range then
            return In_The_Maze.List_Of_Rooms (Number);
        else
            Put_Line ("Error: Room number: " & Natural'Image (Number) &
                      " does not belong to maze");
            return null;
        end if;
    end Room_Of;
end Maze;
```

```
with Wall;
with Room;
with Door;
with Maze;
package Maze_Factory is
    type Object is tagged null record;
    type Pointer is access all Object'Class;

    function Make_Maze (The_Factory : in Object) return Maze.Pointer;
    function Make_Wall (The_Factory : in Object) return Wall.Pointer;
    function Make_Door (The_Factory : in Object;
                        From_Room, To_Room : in Room.Pointer)
                       return Door.Pointer;
    function Make_Room (The_Factory : in Object; Number : in Positive)
                       return Room.Pointer;
end Maze_Factory;
```

```
package body Maze_Factory is

    function Make_Maze (The_Factory : in Object) return Maze.Pointer is
    begin
        return new Maze.Object;
    end Make_Maze;
    function Make_Wall (The_Factory : in Object) return Wall.Pointer is
    begin
        return new Wall.Object;
    end Make_Wall;
    function Make_Door (The_Factory : in Object;
                        From_Room, To_Room : in Room.Pointer)
                       return Door.Pointer is
        Result : Door.Pointer := new Door.Object;
    begin
        Door.Create (Result.all, From_Room, To_Room);
        return Result;
    end Make_Door;
    function Make_Room (The_Factory : in Object; Number : in Positive)
                       return Room.Pointer is
        Result : Room.Pointer := new Room.Object;
    begin
        Room.Create (Result.all, Number);
        return Result;
    end Make_Room;
end Maze_Factory;
```

```
with Room.Bombed;
with Wall.Bombed;
package Maze_Factory.Bombed is
    type Object is new Maze_Factory.Object with null record;
    type Pointer is access all Object'Class;

    -- inherit function make_maze
    -- inherit function make_door
    function Make_Room (The_Factory : in Object; Number : in Positive)
                       return Room.Pointer;
    function Make_Wall (The_Factory : in Object) return Wall.Pointer;
end Maze_Factory.Bombed;
```

```
package body Maze_Factory.Bombed is

    function Make_Room (The_Factory : in Object; Number : in Positive)
                       return Room.Pointer is
        Result : Room.Bombed.Pointer := new Room.Bombed.Object;
    begin
        Room.Bombed.Create (Result.all, Number);
        return Room.Pointer (Result);
    end Make_Room;

    function Make_Wall (The_Factory : in Object) return Wall.Pointer is
    begin
        return new Wall.Bombed.Object;
    end Make_Wall;
end Maze_Factory.Bombed;
```

```
with Map_Site;
package Room is
    type Direction is (North, South, East, West);

    type Object is new Map_Site.Object with private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Room : in out Object);
    procedure Create (The_Room : in out Object; With_The_Number : in Positive);
    procedure Set_Side (Of_The_Room : in out Object'Class;
                        In_Direction : in Direction;
                        To_The_Value : in Map_Site.Pointer);

    function Get_Side (Of_The_Room : in Object'Class;
                       In_Direction : in Direction) return Map_Site.Pointer;
    function Number_Of (The_Room : in Object'Class) return Positive;
    function "=" (Left, Right : in Object'Class) return Boolean;

private
    type Side_Type is array (Direction) of Map_Site.Pointer;
    type Object is new Map_Site.Object with
        record
            Side : Side_Type;
            Number : Natural := 0;
        end record;
end Room;
```

```
with Ada.Text_Io;
package body Room is
    procedure Put_Line (Item : in String) renames Ada.Text_Io.Put_Line;

    procedure Enter (The_Room : in out Object) is
    begin
        Put_Line ("Entered Room Number: " & Natural'Image (The_Room.Number));
    end Enter;
    procedure Create (The_Room : in out Object;
                      With_The_Number : in Positive) is
    begin
        The_Room.Number := With_The_Number;
    end Create;
    procedure Set_Side (Of_The_Room : in out Object'Class;
                        In_Direction : in Direction;
                        To_The_Value : in Map_Site.Pointer) is
    begin
        Of_The_Room.Side (In_Direction) := To_The_Value;
    end Set_Side;
    function Get_Side (Of_The_Room : in Object'Class;
                       In_Direction : in Direction) return Map_Site.Pointer is
    begin
        return Of_The_Room.Side (In_Direction);
    end Get_Side;
    function Number_Of (The_Room : in Object'Class) return Positive is
    begin
        return The_Room.Number;
    end Number_Of;
    function "=" (Left, Right : in Object'Class) return Boolean is
    begin
        return Left.Number = Right.Number;
    end "=";
end Room;
```

```
package Room.Bombed is
    type Object is new Room.Object with private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Room : in out Object);
    procedure Create (The_Room : in out Object; With_The_Number : in Positive);

private
    type Object is new Room.Object with
        record
            Contains_Bomb : Boolean := False;
            Bomb_Has_Discharged : Boolean := False;
        end record;
end Room.Bombed;
```

```
with Ada.Text_Io;
package body Room.Bombed is
    procedure Put_Line (Item : in String) renames Ada.Text_Io.Put_Line;

    procedure Enter (The_Room : in out Object) is
    begin
        Enter (Room.Object (The_Room)); -- downcast and call parent
        if The_Room.Contains_Bomb then
            if The_Room.Bomb_Has_Discharged then
                Put_Line ("room is bombed out!");
            else
                Put_Line ("Bomb could go off any minute");
            end if;
        end if;
    end Enter;

    procedure Create (The_Room : in out Object;
                      With_The_Number : in Positive) is
        Basic_Room : Room.Object;
    begin
        Create (Basic_Room, With_The_Number);
        The_Room := (Basic_Room with
                     Contains_Bomb => (With_The_Number = 2),
                     Bomb_Has_Discharged => False);
    end Create;
end Room.Bombed;
```

```
with Map_Site;
package Wall is
    type Object is new Map_Site.Object with private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Wall : in out Object);

private
    type Object is new Map_Site.Object with null record;
end Wall;
```

```
with Ada.Text_Io;
package body Wall is

    procedure Enter (The_Wall : in out Object) is
    begin
        Ada.Text_Io.Put_Line ("Entered Wall");
    end Enter;
end Wall;
```

```
package Wall.Bombed is
    type Object is new Wall.Object with private;
    type Pointer is access all Object'Class;

    procedure Enter (The_Wall : in out Object);

private
    type Percent_Type is delta 0.01 range 0.0 .. 1.0;
    type Object is new Wall.Object with
        record
            Percent_Damage : Percent_Type := 0.0;
        end record;
end Wall.Bombed;
```

```
with Ada.Text_Io;
package body Wall.Bombed is

    procedure Enter (The_Wall : in out Object) is
    begin
        Enter (Wall.Object (The_Wall)); -- downcast and call parent
        Ada.Text_Io.Put_Line ("Percent damage is: " &
                              Percent_Type’Image (The_Wall.Percent_Damage));
    end Enter;
end Wall.Bombed;
```