Objects

Run time direct instances of classes

© Gunnar Gotshalks

Fields

- The attributes within a class are a template for a collection of **fields** in an object

```
class PERSON
feature
    name : STRING
    sex : GENDER
    age : INTEGER
end
```

a_Person

name
sex
age

© Gunnar Gotshalks

Objects

- Variables with a class for a type
- Must have a name **declared** and the name must be **attached** to the object
- Using an object requires two steps: **declaration** and **creation**

```
p : PERSON        -- declare the name p
create p          -- create and attach object to p

w : PERSON                  -- declare the name w
create w.make("Me")         -- create via a function
```

© Gunnar Gotshalks

Creation Operator

- **create** is akin to **new** in C++ and Java

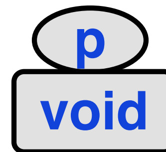
- The 4 steps
  - » **Create an instance of the type**
    - **Allocate enough memory for the instance**
  - » **Initialize each field to default values**
  - » **Attach the reference to the variable**
  - » **Execute the procedure (if any) to complete initialization**

© Gunnar Gotshalks

Reference types

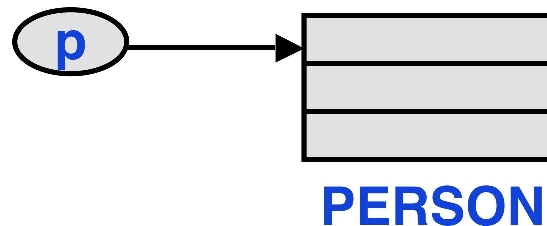
- p is used to **refer to** an instance of type **PERSON**

p : PERSON



- Create and attach object to p – **p is attached**

create p

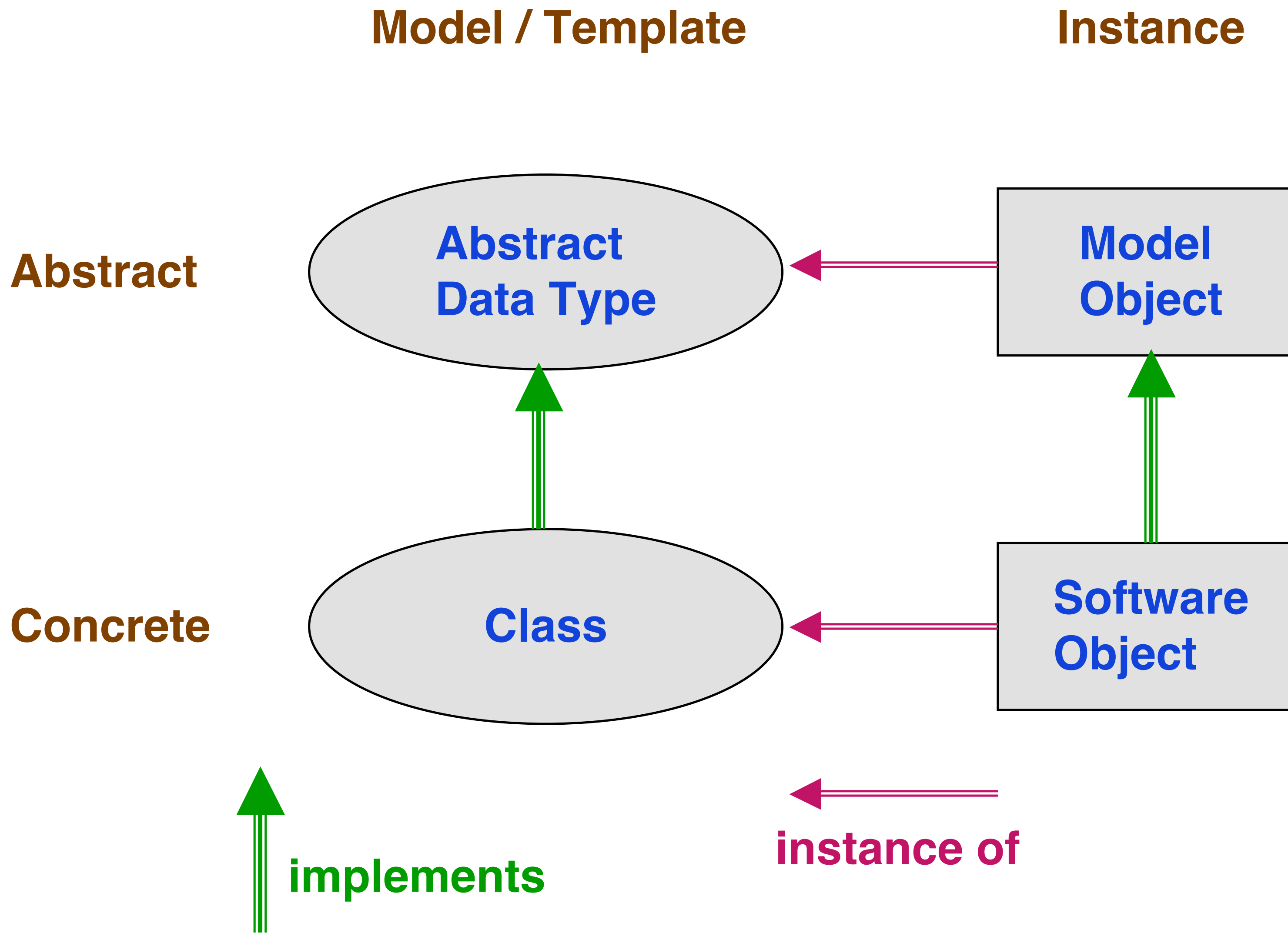


- Think of p as a pointer

For type safety, unlike C/C++, the pointer cannot be de-referenced

© Gunnar Gotshalks

Models & Objects



© Gunnar Gotshalks

Copying

a := y — copies only the reference

a := y.twin

- – shallow copy – one level copy
- – new storage space is created
- – y must exist

a := y.deep_twin

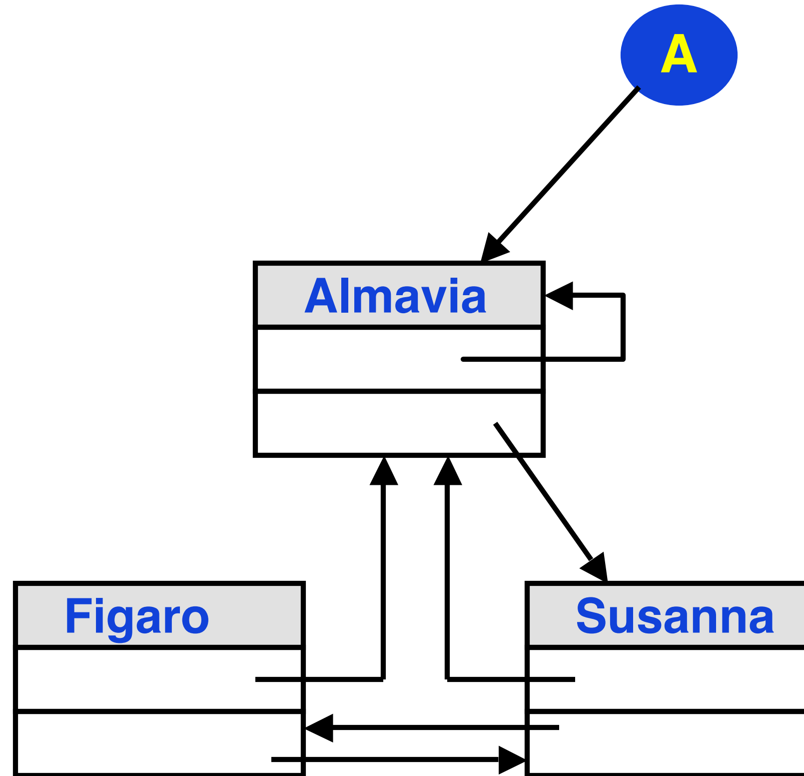
- – deep copy – all levels
- – new storage space is created
- – y must exist

a . copy(y)

- – shallow copy
- – a exists, replace fields of a with those in y
- – NO new storage

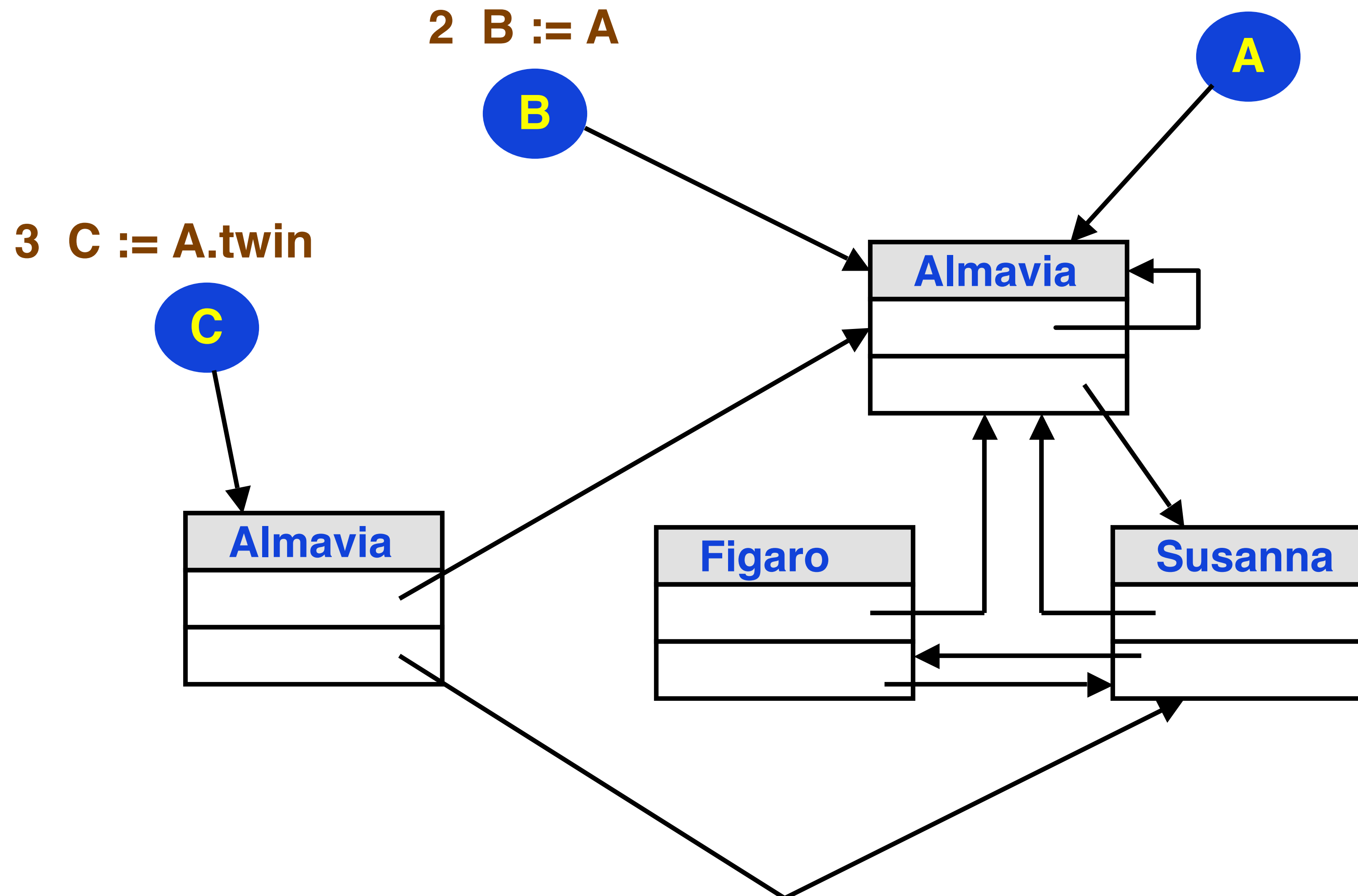
© Gunnar Gotshalks

Copying – 2



© Gunnar Gotshalks

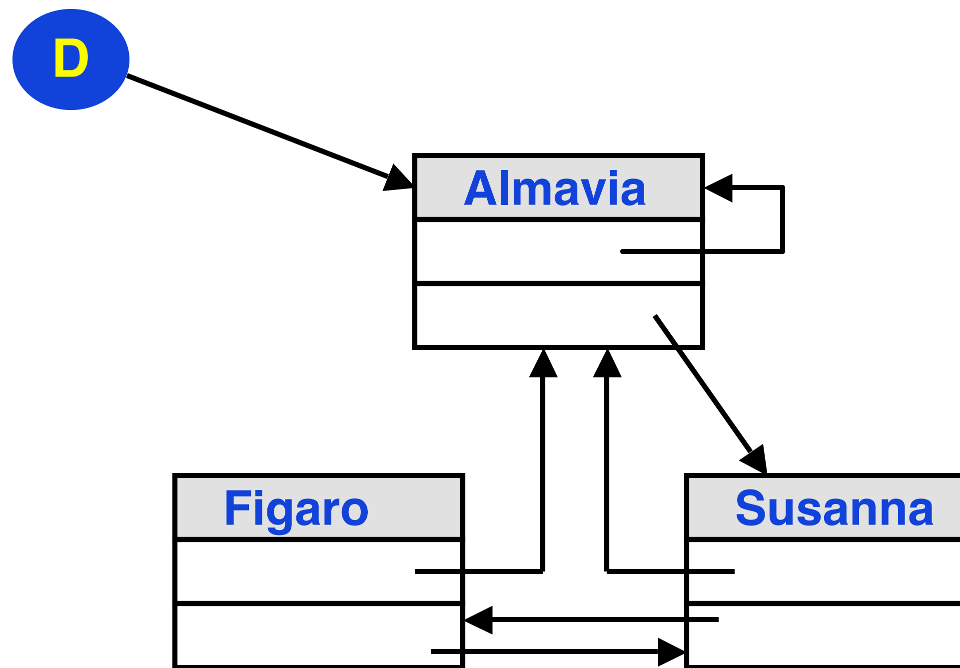
Copying – 3



© Gunnar Gotshalks

Copying – 4

4 D := A.deep_twin -- all new memory locations



© Gunnar Gotshalks

Equality

=	– compares references
equal (a , b)	– shallow comparison – compares one level – works if a is void
a.is_equal (b)	– compares one level – shallow comparison
deep_equal (a , b)	– compares all levels – deep comparison – works if a is void

© Gunnar Gotshalks

Persistence

- Direct dependents
  - **The direct dependents of an object are the objects attached to its references**
- Dependents
  - **The dependents of an object are:**
    - **The object itself**
    - **Its dependents**
    - **And – recursively – the dependents of its direct dependents, etc.**

© Gunnar Gotshalks

Persistence Closure Principle

Whenever a storage mechanism stores an object, it must store with it the dependents of that object.

Whenever a retrieval mechanism retrieves a previously stored object, it must also retrieve any dependent of that object not already retrieved.

© Gunnar Gotshalks

Expanded Types

- In general, declaring a type means the variable is a reference to an instance of the type

- Base type objects – **INTEGER, REAL, DOUBLE, CHAR** – are not referenced

  » **They are statically allocated (expanded)**

  » **They are first class objects – no repackaging as in Java**

- **Expanded** means the reference is replaced with the fields of the referenced object

  » **All usage is the same as for normal OO usage**

© Gunnar Gotshalks

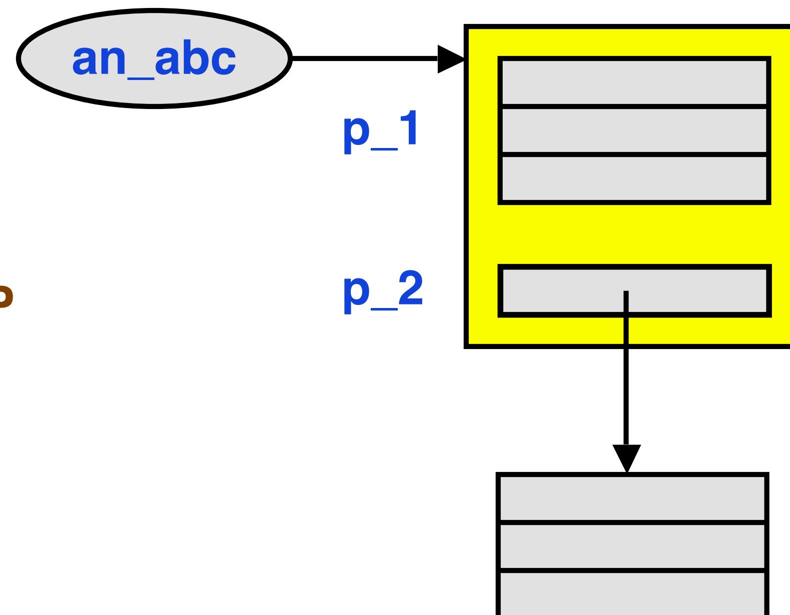
Expanded Types – example

```
class ABC
  p_1 : PERSON_EXP
  p_2 : PERSON
end
```

```
an_abc : ABC
...
print ( an_abc . p_1 . name )
print ( an_abc . p_2 . name )
```

```
class PERSON
  name : STRING
  sex : GENDER
  age : INTEGER
end
```

```
expanded class PERSON_EXP
  name : STRING
  sex : GENDER
  age : INTEGER
end
```



© Gunnar Gotshalks

Composite objects

- An object is called a composite object if one or more of its fields are objects – called **subobjects**

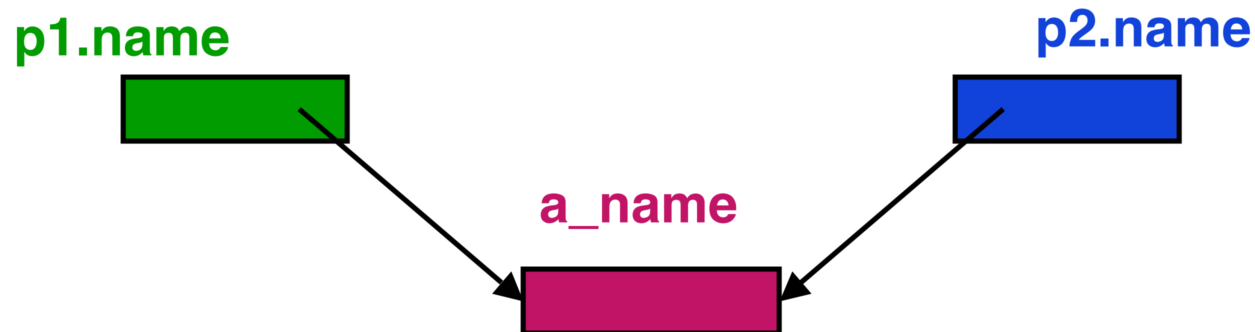
- In the previous slide, the object an_abc is a composite object
  - » **p_1 is an object, not a reference**
  - » **p_2 is a reference, not an object**

- A subobject is a **part_of** an object

© Gunnar Gotshalks

Aliasing

- Occurs when two variables point to the same memory location
- Can lead to surprises but
  - » **Reference assignments needed to benefit from OO**
    - > **Often need two pointers to point to the same object**
  - » **Encapsulation makes it possible to avoid dangers of reference manipulations**



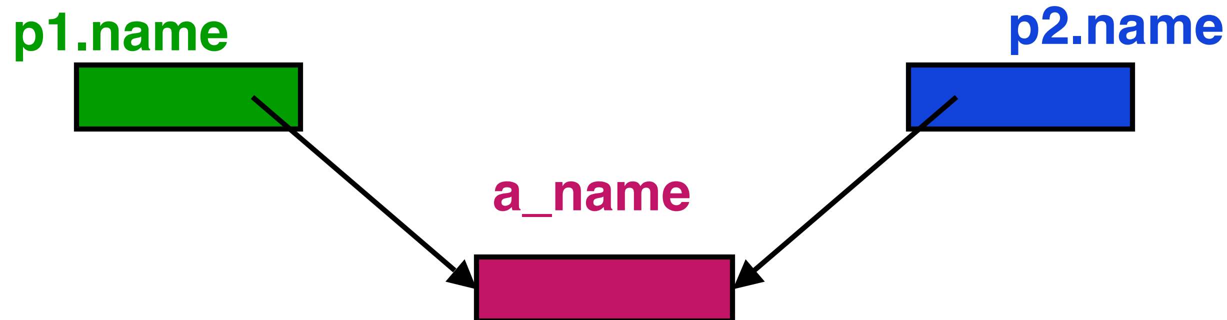
© Gunnar Gotshalks

Aliasing Potential Problem

- Consider the following version of class Person

```
class PERSON
feature
    name : STRING
end
```

- We say that Person **has a** NAME
- NAME is a reference
  - » **Makes it possible for two or more instances of PERSON to share the same name**



© Gunnar Gotshalks

Aliasing Potential Problem – 2

- Sharing references can lead to surprises
  - » **p1.name := "John" ; p2.name := p1.name**
  - » **print (p2.name) --> John**
  - » **p2.name.put('x' , 1)**
  - » **print (p1.name) --> xohn** **-- Probably a surprise**
- Should be careful to have names pointing to different memory locations
  - » **condition: p1.name ≠ p2.name**
  - » **But could be difficult to enforce across objects**

© Gunnar Gotshalks

Name Solution

- Could use expanded classes
  - » **But unless planned for cannot use existing classes**
    - > **E.g. STRING in Eiffel**
- Can use twin or deep_twin, as appropriate, and contracts

```
class PERSON

make (the_name: STRING)
 require good_name: the_name.count > 3
 do name := the_name.twin
 ensure
  name_set: name /= the_name  and  name . is_equal (the_name)
 end

name: STRING
end
```

© Gunnar Gotshalks