Static and Dynamic Behavior

לאוניד ברנובים וורג'ワイס
המחלקה למודע המחשב
אוניברסיטת בן-גוריון
Roadmap

In this chapter we will examine how differences in static and dynamic features effect object-oriented programming languages:

- Static versus Dynamic Typing
- Static and Dynamic Classes in Statically Typed Languages
- Static and Dynamic Method Binding in Statically Typed Languages
What do the terms Static and Dynamic Mean?

In programming languages:

- **Static** almost always means fixed or bound at **compile time**, and cannot thereafter be changed.

- **Dynamic** almost always means not fixed or bound until **run time**, and therefore can change during the course of execution.
Static and Dynamic Typing

In a statically typed programming language (e.g., Java or Pascal), for example, variables have declared typed -- **fixed at compile time**

In a dynamically typed programming language (e.g., Smalltalk or LISP), a variable is just a name. Types are associated with values, not variables. **A variable can hold different types during the course of execution**
Arguments for and Against

- **Static typing** allows better error detection, more work at compile time and hence faster execution time.

- **Dynamic typing** allows greater flexibility, easier to write (for example, no declaration statements).
Both arguments have some validity, and hence both types of languages will continue to exist in the future.
The Polymorphic Variable

The addition of object-oriented ideas in a statically typed languages adds a new twist

Recall the argument for substitution: an instance of a child class should be allowed to be assigned to a variable of the parent class:

```plaintext
var
    pet   : Mammal;
    fido : Dog;
    felice: Cat;
begin
    pet  := fido; // legal
    pet  := felice; // legal
    fido := pet;  // not legal
```
Static Class and Dynamic Class

- In a statically typed OO language we say the class of the declaration is the **static class** for the variable.

- While the class of the value it currently holds is the **dynamic class**.

- Most statically typed OO languages constrain the dynamic class to be a child class of the static class.

```plaintext
var
    pet : Mammal;
    fido : Dog
begin
    pet := fido; // static class is Mammal, dynamic class is Dog
end;
```
Importance of Static Class

- In a statically typed object-oriented language, the **legality of a message** is determined at compile time, **based on the static class**.

- A message can produce a compile error, even if no run-time error could possibly arise:

```java
class Mammal { }
class Dog extends Mammal {
    void speak() { System.out.println("woof"); }
}
Mammal pet = new Dog();
pet.speak(); // will generate error, Mammals don't speak
```
Run Time Determination

• The principle of substitution can be viewed as moving a value up the inheritance hierarchy.

• Occasionally it is necessary to do the reverse:

To determine if a value currently being held by a variable declared with one class type is, in fact, derived from a class that is lower in that class hierarchy.

• For example:

To determine if a variable of type Animal is, in fact, a Dog.
Reverse Polymorphism / Down Casting

Polymorphism says we can assign a value from a child class to an instance of the parent class, but can this assignment then be reversed? Under what conditions?

```pascal
var
    pet    : Mammal;
    fido   : Dog;
    felice : Cat;

begin
    pet := fido; // legal
    fido := pet; // is this legal?
end;
```

This is known as the problem of reverse polymorphism
Two aspects of reverse polymorphism

There are two specific problems associated with the question of reverse polymorphism.

- **The problem of identity** - can I tell if a value declared as an instance of a parent class actually holds a value from a subclass

- **The task of assignment** - can I then assign the value from the parent class to a variable declared as the subclass

In some languages mechanisms are provided to address these two problems together, while in other languages they are separated.
Run-time testing without language support

class Animal {
    public:
    virtual Dog * isDog() {return NULL;}
    virtual Bird * isBird() {return NULL;}
    ...
};

class Dog : public Animal {
    virtual Dog * isDog() {return this; }
    ...
};
Testing Message Understanding

- In dynamically typed OO languages it is generally true that the legality of a message passing expression cannot be known until run-time.

- If the receiver does not know how to respond to a message, a run-time exception is thrown.

- Occasionally, it is useful for the programmer to be able to test whether a receiver will understand a particular message before actually attempting to pass the message to the receiver.

- Languages provide different mechanisms to do this:
  
  **Smalltalk**: `(aPet respondsTo: #bark) ifTrue: [...]`
  
  **Ruby**: `if aPet.respond_to?(“bark”)` ...
  
  **Objective-C**: `if ( [aPet respondsToSelector: @selector(bark)] ) { ... }`
The Container Problem

The task of reverse polymorphism is often encountered in connection with a collection of values:

- We have a list of items from the parent class (say a list of Mammals), and when we extract a value we need to know if it is a more specific type.

- Generally occurs in languages with a single inheritance tree, where the only type we may have associated with a value is the class “Object”.

- Solving this problem generally requires values to have “self knowledge” of their own type.

- In some languages they do, in some languages values do not.
Static and Dynamic Method Binding

Should the binding for information be associated with the static class of a variable or the dynamic class?

Alice holds a small Mammal - asks Bill “does this animal give birth to live young”:

- **Static answer**: All mammals give birth to live young - therefore **yes**
- **What if the Mammal is a platypus?**
- **Dynamic answer**: Platypus lay eggs, therefore **no**

Even statically typed OOP languages can use dynamic binding. But may use static type to determine legality of operation.
Documenting Method Binding

- In many languages dynamic binding is the default. If a child class overrides a method in the parent, using the same type signature, then the selected method will be determined by the dynamic type.

- In other languages (C++, Delphi, C#) the programmer must indicate which methods are dynamically bound and which are statically type. In C#, for example, this is done using the `virtual` and `override` keywords.

```cpp
class Animal {
public:
    virtual void speak () { cout << "Animal Speak !\n"; }
    void reply () { cout << "Animal Reply !\n"; }
};

class Dog : Animal {
public:
    override void speak () { cout << "woof !\n"; }
    void reply () { cout << "woof again!\n"; }
};

class Bird : Animal {
public:
    override void speak () { cout << "tweet !\n"; }
};
```

Animal a;
Dog b;
b.speak();
woof !
a = b;
a.speak();
woof !

Bird c;
c.speak();
tweet !
a = c;
a.speak();
tweet !
Method Binding in C++

C++ is the most complex language. Not only must the programmer use the virtual keyword, but true polymorphism only occurs with pointer or reference variables.

```cpp
class Animal {
public:
    virtual void speak () { cout << "Animal Speak !\n"; }
    void reply () { cout << "Animal Reply !\n"; }
};

class Dog : public Animal {
public:
    virtual void speak () { cout << "woof !\n"; }
    void reply () { cout << "woof again!\n"; }
};

class Bird : public Animal {
public:
    virtual void speak () { cout << "tweet !\n"; }
};

Animal * a;
Dog * b = new Dog();
b->speak();
    woof !
a = b;
a->speak();
    woof !
Bird c = new Bird();
c->speak();
tweet !
a = c;
a->speak();
tweet !
```

We will see an explanation for the curious C++ semantics when we discuss memory management in the next chapter.
Merits of Static versus Dynamic Method Binding

Arguments concerning static versus dynamic binding mirror those concerning static versus dynamic typing:

- **Efficiency** - static binding uses least CPU cycles, dynamic binding requires more time

- **Error detection** - static binding permits errors to be caught at compile time rather than run-time

- **Flexibility** - dynamic binding permits greater flexibility, static binding creates rigidity and inhibits reuse
Chapter Summary

- A statically typed language associated types with variables, a dynamically typed language associates types with values.
- Static typing gives better error detection, better run-time efficiency, less flexibility.
- In a statically typed OO language, an object variable can still hold values from a child class.
- The static class is the class of the declaration, the dynamic class is the class of the value currently held.
- The legality of a message is checked using the static class.
- A message can be bound to a method using either the static or dynamic class.
- Most languages use the dynamic class. Some languages allow the programmer to choose which method is used.