Introduction to
Object Oriented Programming 2E
Timothy A. Budd

Chapter 4

Messages, Instances, and Initialization

Topics to be Covered

Major topics covered in this chapter:

- Instances
- Message Passing (syntax, not meaning)
- Creation and Initialization

Creation of New Variables

How is space created for new variables?

Static allocation: using declaration statements – allocated before program begins execution (Fortran, C global variables)

Automatic allocation: using declaration statements – automatically allocated on entering scope – stack-based storage management (Pascal local variables)

Dynamic allocation: using explicit directives – may not be automatic – heap-based storage management

Storage Management Approach

explicit management by programmer: programmer must return unneeded memory to system – C++ `delete`, Object Pascal `dispose`

Advantage: efficiency

Disadvantages:

- attempted use of memory not yet allocated or already freed
- multiple freeing of memory
- memory leak – allocated memory is never released

implicit management by runtime system: system detects when data unneeded, automatically recovers memory – garbage collection – Smalltalk, Java

Advantage: safety and convenience

Disadvantage: relative inefficiency

Syntax for Automatic Creation

In C++ and Object Pascal (and sometimes in Objective-C) memory for variables is often created using a simple declaration statement.

```
Card aCard;    // C++ declaration

var
    aCard : Card;    { Object-Pascal declaration }
```

Syntax for Dynamic Creation

Always in Smalltalk and sometimes in other languages, objects are created dynamically using the “new” operator.

```
d ← Card new.

aCard = new Card;

aCard = new Card(Card.spade,5);
```

Message Passing

Message passing differs from a simple function call:

- Message always transmitted *to* specific instance—the receiver
- Instance's class decides particular method invoked
- When method invoked, receiver passed as implicit extra parameter
- Receiver accessible using pseudo-variable **self** in Smalltalk and Object Pascal, **this** in C++ and Java

Message Passing Syntax

Two major forms - receiver, then either period or space, then arguments:

```
aCard.flip();

aCard.includes(a, b);

aCard flip.

aCard includes: a and: b.
```

Operators as Messages

- Smalltalk and C++ operators can be defined as methods.
  - left argument is receiver
  - right argument is method's single parameter

```
a + b
```

- Java does not support "operator overloading"

Creation and Initialization

Many OOP languages (Smalltalk, C++, Java, Objective-C) tie together object *creation* and *initialization*.

- Avoids creating an object without properly initializing its fields
- Avoid creating an object and initializing its fields more than once

We discuss two mechanisms, classes (factory) methods and constructors.

Class Methods

- Smalltalk and Object Pascal use "object creators" to initialize objects at creation
- Object creators associated with class as whole, not an instance

```
id aCard;

aCard = [ Card rank: 2 suit: 3 ];
```


Constructors in Java

As in C++, a constructor in Java is a method (i.e., member function)

- has same name as the class
- can take argument lists, like other functions.
- invoked implicitly by `new`, explicitly by `this` in other constructors

```
class newClass {
    ...
    newClass (int i) {
        ... // initialization
        }
}
```

Destructors in Java

- Garbage collection recovers memory no longer referenced
- Method `finalize()`, if defined, called when reclaiming memory for object
  - Does any other “housecleaning” needed (close files, etc.)
  - Executes at unpredictable time
  - Should not be depended upon for correctness – do critical “housecleaning” tasks explicitly