Modern Programming Languages

C++

* Classes
* Inheritance multiple inheritance
* Method and operator overloading
* Templates
* Default parameters
* New standard library

Few syntactic differences from C.

Classes in C++

Structs were used in C, almost like classes

Typedef struct {

Int layers;

Float price;

Char \*baker\_name;

} Cake;

Cake cake1 = {2, 14.5, “John};

Printf(“Layers %d”, cake1.layers};

C++ introduced class

Class classname {

// Instance variables and methods

// Members private by default

};

class Cake {

public:

int layers;

float price;

string baker\_name;

};

Cake cake1;

Cake1.baker\_name = “Owen”;

Cake1.layers = 4;

Cake1.price = 35.50;

Cout << cake1.baker\_name << endl;

# In object oriented programming, variables should be private and only accessed via getters and setters.

## Methods

Void: Does not return anything

Return\_type [class\_name::]method\_name(parameters …) {

// method body

}

In C++ methods can be defined **inside or outside** the class declaration.

Class Cake {

Public:

Int layers;

Float price;

String baker\_name;

Void vake(int temp) {

Cout << baker\_name << “ is baking the cake at “ << temp << “C.” << endl;

}

};

void Cake::icing() {

cout << “Method defined outside the class” << endl;

}

Cake cake1;

Cake1.baker\_name = “Owen”;

Cake1.bake(200);

## Constructors & Destructors

* Special purpose methods to initialise or destroy an object
* Constructor: same as Java, method with the same name as the class, no return type.
* You can have as many constructors as you like.

Class Cake {

Int layers;

Float price;

String baker\_name;

Public:

Cake() {layers = 1; price = 10.0; baker\_name = “Unknown”; }

Cake(int p) {price = p; \_etc }

}

Destructors:

* Called automatically whenever an object is destroyed
* For local variables when they go out of scope.
* Only needed in classes that have a pointer to dynamically allocated memory.
* Method with name of class preceded by ~

Class array {

Double \*a;

Int len;

Public:

array(int l = 0) {len = l; }

~array() { delete[] a; }

};

delete[] frees the memory.

## Access Controls

Private – only used by member methods and friends of the class

Public – Used by member methods, friends and classes derived from the class

Protected – Can be used by any method

Usually instance variables are declared in the private section, and the methods in the public sect.

C++ Friends

Classes, methods can be declared as friends. They are granted access to private methods in a class.

## Inheritance

Class derived\_class: access\_modifier base\_class {

// class body

};

Subclassing process can be declared with access controls.

Method overriding is permitted.

Class Cake {

Int layers;

Float price;

String baker\_name;

Protected:

Float profit;

Public:

Int calories;

};

class BirthdayCake : public Cake {

// in this class, profit and calories are private, and no derived class has access to any member to cake class.

};

## Virtual Methods

Object of a derived class can be referred to using a pointer or reference to a base class type.

A C++ method can be defined to be virtual, meaning dynamically bound (run-time).

Early compiler according to declared type of pointer/reference

Late- runtime – according to actual type being referred to.

Abstract Classes:

* Class that is too general to create real objects
* Used to provide an implementation guide for other classes.
* Many inheritance hierarchies have abstract superclasses occupying the top few levels.

C++ realises abstract classes though pure virtual functions.

Usually top layers are abstract classes.

Class Food {

Public:

Virtual void eat() = 0;

// pure virtual function indicated by “=0”

};

## Multiple inheritance

Class BirthdayCake : public Cake, public LuxuryItem {

// class definition as usual

// data members, methods

};

If there are two inherited members with the same name, they can both be referenced using the scope operator (::)

## Templates

Templates introduce generic programming into C++

Act as a blueprint for creating a class, method independent of any particular type.

Three kinds of C++ templates:

* Method templates
* Class templates
* Variable templates

Parametric polymorphism

Parameterize a variable – e.g. by calling a type on a variable, you can call it in different forms.

Templates look very similar to classes.

Java

## WORA- Write Once Run Anywhere

Source is compiled into byte code, which can be run anywhere.

Interpreted languages are slower than compiled ones, but it trades speed for platform independence and programmer safety.

I.e. creates a class file that is actually run.

Java compilers are good.

## Java- Basics

Heavily inspired by C++ and smalltalk

Public class Cake {

Private int layers;

Public Cake() {

Layers = 1;

}

public Cake(String name) {

layers = 1;

}

public void bake(int temp) {

System.out.println(bakerName + “is baking at”);

}

}

You can overload stuff with two Cake methods.

Cake cake1 = new Cake();

Cake1.bake(250);

## Basics control

Almost everything is an object.

Only primitive types are not objects.

Method arguments are always passed by value.

Method overloading – but no operator overloading.

No structs.

Objects are not copied – only their references are.

Inherently multi-threaded

Powerful and easy-to-use libraries for data structures, multi-threading, networking, I/O, graphics.

## Access controls

Public – visible to all classes everywhere

Default – Class with no access modifier declaration is visible within its own package.

Protected – Accessible by classes within my own package and classes derived from the class.

Private – Can only be accessed in own class.

Every Java app must contain a main method whose signature is:

public static void main(String[] args)

public main class must have the same name as a file.

System.out.println(“Hello world”);

Class called system, with a class variable out with a method called println.

## Constructors

Foo f = new Foo();

The object referenced by f is allocated in the heap (heap-dynamic???)

Constructors – no return, method with the same name as the class.

No destructors!!

Classes can define data members that are associated with the class, instead of each object.

Static keyword is used.

E.g. in the constructor you can increment a static variable, tallying up how many instances there are.

A static method can be invoked directly through the class without invoking a new instance of the class.

Only single inheritance is supported in Java.

Method overriding is permitted **if they are not final**.

Super keyword can be used to invoke parent class method/constructor.

## Inheritance

Abstract classes help reduce code dependencies.

Java only has single inheritance.

A special category of class provides some of the benefits of multiple inheritance.

* An interface can include only certain method declarations and named constants.
* A class implements an interface.
* **An interface can extend another interface!**
* Now interfaces can have default or static methods, which is bad for complexity.

Public interface Bakeable extends edible {

Void bake();

}

public class Cake implements Bakeable {

public void bake() {

// Must implement this behaviour

}

}

**Generics**

Generics enable types(classes and interfaces) to be parameters when defining classes, interfaces and methods.

Using generics, programmers can implement generic algorithms that work on collections of different types.

I.e. a method that takes in an object that returns a type of object.

A generic class that holds an object.

Defining a class as a container for an object.

## Summary

Design decisions to support OOP are similar to C++

Doesn’t support procedural programming. -> Everything has to be in a class.

No parentless classes.

Interfaces provide a simple form of support for multiple inheritance.

Polymorphism- ad hoc, subtyping, parametric

Nested classes

Dynamic binding is used as “normal” way to bind method calls to method definitions.

## Ruby

Everything is an object

All computation is through message passing.

A lot of stuff from SmallTalk.

Class definitions are executable, as well as method definitions.

Some access control.

You can do operator overloading, but not method overloading.

All variables are typeless, so method overloading does not make sense.

PROLOG BITCH

Syntax

<- becomes :-

“,” is used for logical “and”

Clauses end with “.”

To execute a program, you must make a query.

Dynamically type- no type declaration

A single datatype – *term*

**Types of terms:**

* Atoms
* Numbers
* Variables
* Compound terms

**Atoms**- primitive data item. Begins with lowercase, contains spaces.

**Numbers** – integers, float etc

**Variables** – Begins with a capital letter (A-Z) or “\_”

Once a variable gets a value, there is no way to change it.

No destructive assignments.

Variable can be aliased – sharing their values with other variables.

Variable is instantiated when it has a value, otherwise it is uninstatntiated.

**Compound terms** – consist of FunctionName(Subterm,… Subterm)

functorName must be an atom.

Subterm is a constant, variable or another nested term.

E.g. has(owen, jaguar)

Date(9, Monthm 2017)

President(usa, Trump)

Compount term: functor and arguments

Arity –how many arguments there are.

Terms are stored as trees.

Matching terms- ie. Compared

Match if they are identical or if their variables can be assigned values.

Terms are matched by: traversing their trees, comparing their nodes and assigning values to variables – if needed.

Does date(2, april, Year) match date(Day, Month, 2017)?

Subterms in a functor are **ordered.**