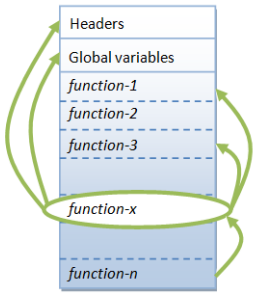
Introduction

Procedural languages (such as C, Fortran, Cobol and Pascal) suffer some notable drawbacks in creating reusable software components:

- The procedural programs are made up of functions. Functions are less reusable. It is very difficult to copy a function from one program and reuse in another program because the function is likely to reference the global variables and other functions. In other words, functions are not well-encapsulated as a self-contained reusable unit.
- The procedural languages are not suitable of high-level abstraction for solving real life problems. For example, C programs uses constructs such as if-else, for-loop, array, method, pointer, which are low-level and hard to abstract real problems such as a Customer Relationship Management (CRM) system or a computer soccer game.



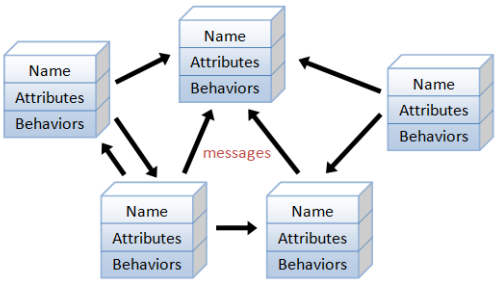
A function (in C) is not well-encapsulated

Object-oriented programming (OOP) languages are designed to overcome these problems.

Object-Oriented Programming (OOP)

The basic unit of OOP is a `class`, which encapsulates both the static properties and dynamic operations within a "box", and specifies the public interface for using these boxes. Since classes are well-encapsulated, it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.

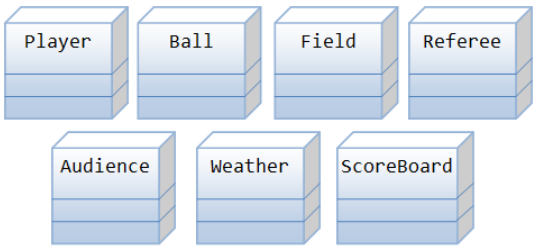
OOP languages permit higher level of abstraction for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++ and C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

As an example, suppose you wish to write a computer soccer game (considered as a complex application). It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" in the soccer game. For example Player entity can be elaborated as following.

- `Player:` attributes include name, number, location in the field, and etc; operations include run, jump, kick-the-ball, and etc.



Classes (Entities) in a Computer Soccer Game

Similarly the other entities can have their own attributes and behaviour. Most importantly, some of these classes (such as Ball and Audience) can be reused in another application, e.g., computer basketball game, with little or no modification.

OOPs Concepts

The basic OOPs concepts are:

1. Class
2. Objects
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

Class

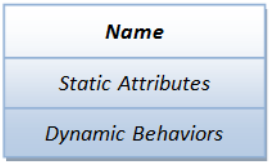
In Java, a class is a definition of objects of the same kind. In other words, a class is a blueprint, template, or prototype that defines and describes the static attributes and dynamic behaviors common to all objects of the same kind.

An instance is a realization of a particular item of a class. In other words, an instance is an instantiation of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".

The term "object" usually refers to instance. But it is often used loosely, which may refer to a class or an instance.

A Class is a 3-Compartment Box encapsulating Data and Operations

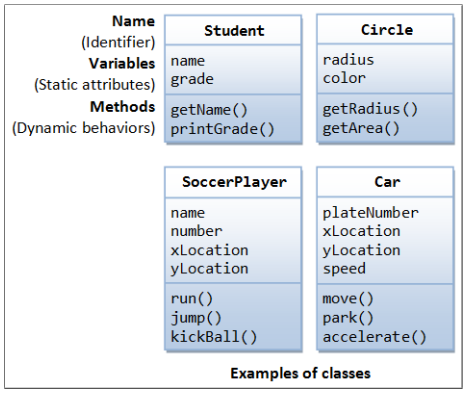
A class can be visualized as a three-compartment box, as illustrated:

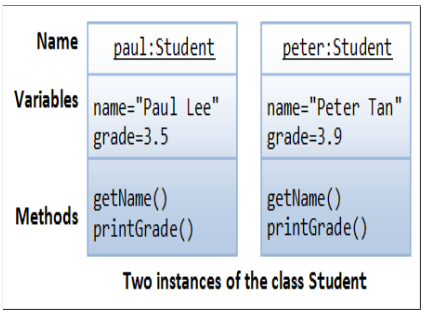


A class is a 3-compartment box

1. `Name (or identity):` identifies the class.
2. `Variables (or attribute, state, field):` contains the static attributes of the class.
3. `Methods (or behaviors, function, operation):` contains the dynamic behaviors of the class.

The following figure shows a few examples of classes and two instances of the class `Student`, identified as "`paul`" and "`peter`".





Syntax:

```
[AccessControlModifier] class ClassName {
    // class body contains definition of variables and methods
    ...
}
```

Class Definition

In Java, we use the keyword `class` to define a class. For examples:

```
public class Circle {
    // variables
    double radius;
    String color;
    // methods
    double getRadius() {...}
    double getArea() {...}
}
```

Objects

A thing in a real world that can be either physical or conceptual. An object in object oriented programming can be physical or conceptual.

Conceptual objects are entities that are not tangible in the way real world physical objects are. Bulb is a physical object. While college is a conceptual object. Conceptual objects may not have a real world equivalent. For instance, a Stack object in a program.

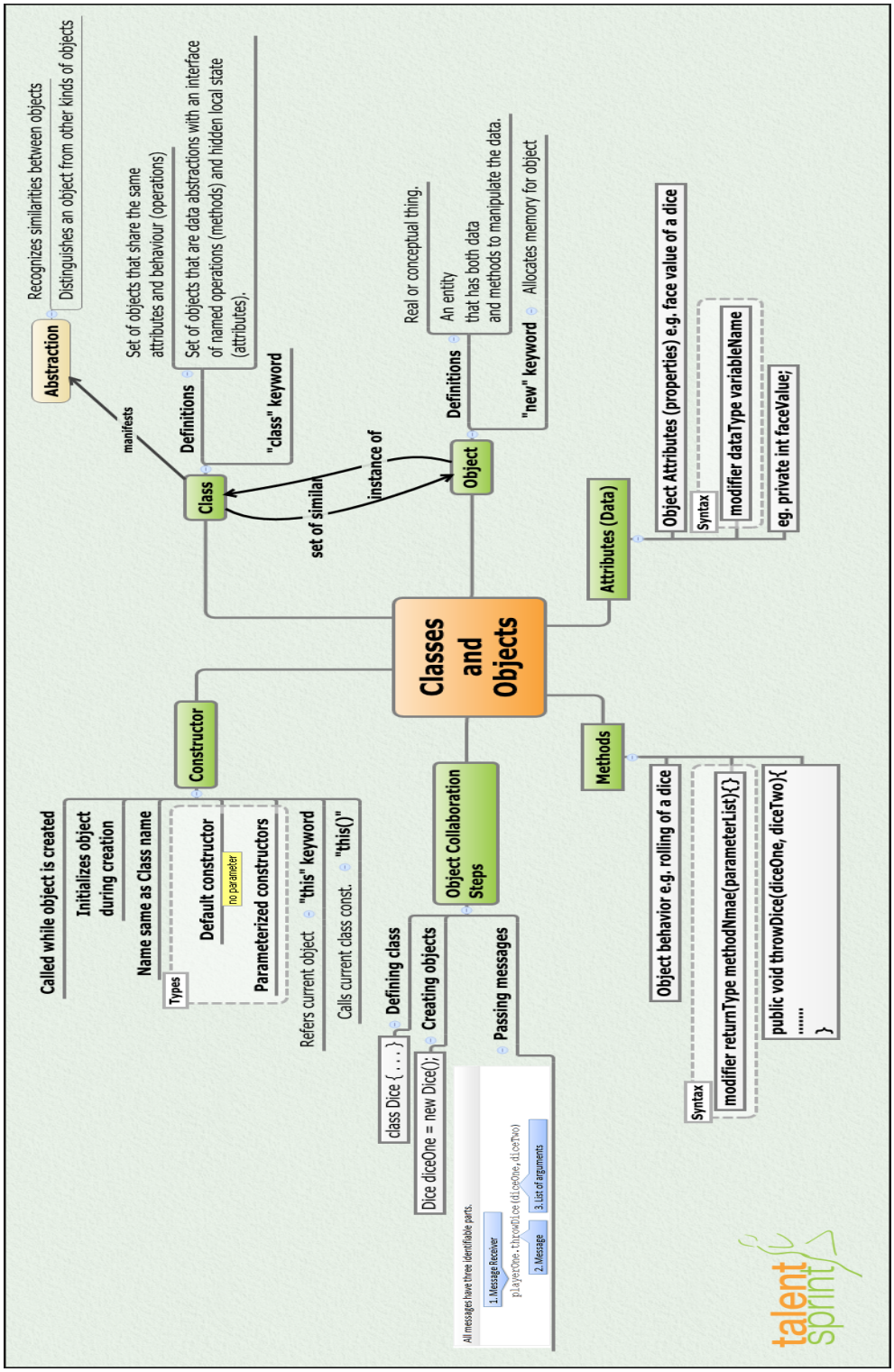
Object has state and behavior.

To create an instance of a class, you have to:

1. Declare an instance identifier (instance name) of a particular class.
2. Construct the instance (i.e., allocate storage for the instance and initialize the instance) using the "`new`" operator.

For example, suppose that we have a class called `Circle`, we can create instances of `Circle` as follows:

```
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;
// Allocate and construct the instances via new operator
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");
// You can declare and construct in the same statement
Circle c4 = new Circle();
```

Abstraction

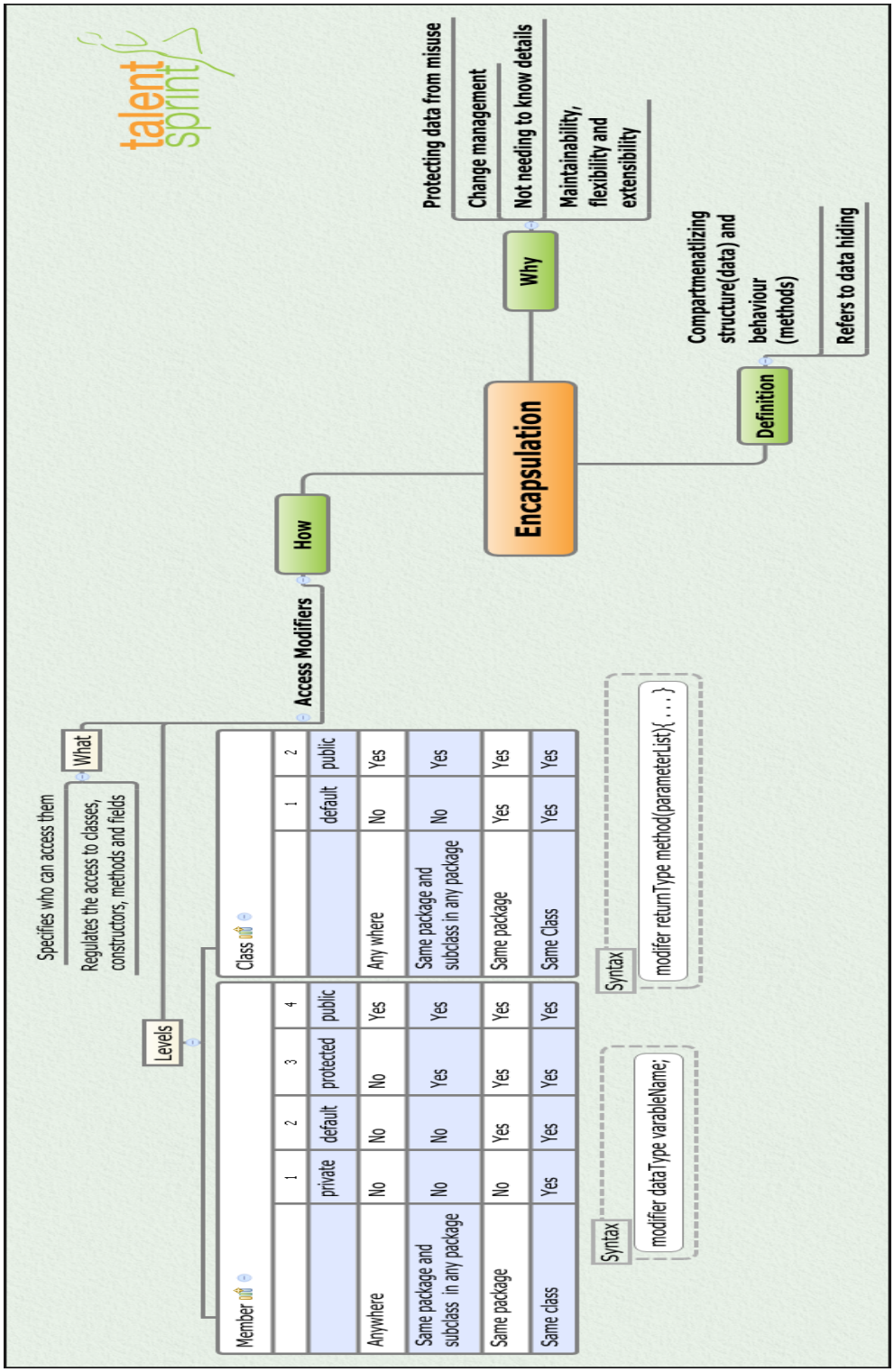
One of the chief advantages of object oriented programming is the idea that programmers can essentially focus on the "big picture" and ignore specific details regarding the inner-workings of an object. This concept is called abstraction.

For example a car in itself is a well-defined object, which is composed of several other smaller objects like a gearing system, steering mechanism, engine, which are again have their own subsystems. But for humans car is a one single object, which can be managed by the help of its subsystems, even if their inner details are unknown.

Encapsulation

- Abstraction in OOP is closely related to a concept called encapsulation.
- The Object Orientation has two major promises/benefits. They are: flexibility and maintainability.
- You have to write your classes and code in a way that supports flexibility and maintainability.
- The ability to make changes in your implementation code without breaking the code of others who use your code is a key benefit of encapsulation.
- If you want maintainability, flexibility and extensibility your design must include encapsulation.
- Encapsulation is the mechanism that binds together the code and the data it manipulates, and keeps both safe from outside interference and misuse.
- The following are some of the ways to include encapsulation:
  - Keep instance variables protected (with an access modifier, often private).
  - Make public accessor methods, and force calling code to use those methods rather than directly accessing the instance variable.
  - For the methods, use the JavaBeans naming convention or follow lowerCamelCase convention

    set<someProperty> and get<someProperty> or isStudentApproved()



Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

Inheritance

One of the main concept of OOP is inheritance.

A child class inherits properties and attributes from its parents.

Inheritance is the process by which one class acquires the properties of another class. By use of inheritance, a class needs only to define all of its characteristics that make it unique within its class; it can inherit its general attributes from its parent.

The derived class contains all the attributes and behaviors of its parent class. Moreover the derived class can define its own attributes and behaviors.

The derived class can override the definition of existing methods by providing its own implementation.

The code of the derived class consists only the changes and additions to the base class.

Need of Inheritance

Inheritance is required for the following reasons:

Modular coding, which means less code and easier to get an idea about the code

- Code reuse
- Do not break what is already working
- Easier updates

For example, Car is a classification of Four Wheeler. Here Car acquires the properties of a four-wheeler. Other classifications could be a jeep, tempo, van etc. Four Wheeler defines a class of vehicles that have four wheels, and specific range of engine power, load carrying capacity etc. Car (termed as a sub-class) acquires these properties from Four Wheeler (termed as a super-class), and has some specific properties, which are different from other classifications of Four Wheeler, such as luxury, comfort, shape, size, usage etc.

A car can have further classification such as an open car, small car, big car etc, which will acquire the properties from both Four Wheeler and Car, but will still have some specific properties. This way the level of hierarchy can be extended to any level.

Java Swing and Awt classes represent best examples for inheritance.

Polymorphism

Polymorphism (from Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions that is one interface with multiple methods.

Generally, polymorphism allows you to mix methods and objects of different types in a consistent way.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car.

Polymorphism could be static and dynamic both. **Overloading** is static polymorphism, where, **overriding** is dynamic polymorphism.

Method Overloading

In this case, different methods within the same class or in a common hierarchy share the same name but have different method signatures (name + parameters).

```
public static float max(float a, float b)
public static float max(float a, float b, float c)
public static int max(int a, int b)
```

When a method is called, the call signature is matched to the correct method version.

Note: This is done during program compilation

Object class

Every class in Java extends class **Object** which is stored in the **java.lang package** and is the ultimate superclass of all Java classes (except for Object).

Any class that does not explicitly extend another class, implicitly extends `Object` class.

Few important methods of the `Object` class are as follows:

equals(Object obj): Indicates whether some other object is "equal to" this one and it returns a boolean value as true or false

toString(): Returns a string representation of the object

hashCode(): Returns a hash code value (of integer type) for the object

Instance and Local Variables

Instance (or member) Variables

- Accessible anywhere in the class
- Automatically initialized before invoking any constructor
- Static variables are initialized at class load time
- Can have the same name as the class
- This differentiates one object from another, giving an object its individuality. For example, the particular `name` and address for a given `Person` object is declared as an instance variable.
- An instance variable relates to an instance (object) of its class.

```
class Person {
    private String name;
    public void setName(String aName) {
        name = aName;
    }
    public String getName() {
        return name;
    }
}
```

Here, instance variable called `name` of type `String` declared inside the class `Person`.

Local Variables

- Must be initialized explicitly and (Or, the compiler will catch it.) Object references can be initialized to `null`.
- The following code will not compile. Specify else part or initialize the local variable explicitly.

```
public String testMethod(int a) {
    String tmp;
    if (a > 0)
```



```
            tmp = ''Positive'';
        return tmp;
    }
```

Can have the same name as a member variable and resolution is based on scope.

