Object Oriented Programming

What are object oriented concepts? What is difference between object-based, object-oriented and fully object-oriented language?

- Pillars of Object Oriented Programming ... Grady Booch (Object Oriented Analysis Design)
  - Major Pillars ... Must be supported by any Object Oriented language
    - Abstraction
    - Encapsulation
    - Hierarchy ... Reusability
      - is-a (inheritance)
        - Person <|--- Employee <|--- Manager
      - has-a (association)
        - Computer <>--- Motherboard <>--- Processor
      - use-a (dependency)
        - MyLinkedList "use a" Stack class for reversing list.
      - creates-a (instantiation)
        - SessionFactory creates Session.
    - Modularity
      - Divide source code into multiple logical units ... better maintainability
      - Division
        - Source code files
        - Classes
        - Packages, Namespaces, Modules
  - Minor Pillars ... May be supported by any Object Oriented language
    - Typing
      - Strong vs Weak -- Data types
        - Java: Strong typing
        - JS: Weak typing
      - Static vs Dynamic -- Polymorphism

        - Compile time
        - Runtime
    - Concurrency
        - Object behaviour in case of multi threading (how it work with race condition?)
    - Persistence
        - Maintaining state of object outside its scope (e.g. database, file - serialization, network)
- Object Based
    - Supports only abstraction & encapsulation (i.e. classes/objects)
    - e.g. Visual Basic, Old Java script, ...
- Object Oriented
    - Must support all major Pillars
    - e.g. C++, Java
- Fully Object Oriented
    - Must support all major and minor Pillars
    - Each data must be object (even primitive types must be objects)
    - No global variables/functions
    - e.g. C#

What is abstraction and encapsulation. Give real-life example.

- Abstraction
    - Getting essential details of the system.
    - Depends on perspective of user.
    - Outer view of the system.
- Encapsulation
    - Binding code and data together in a class.
    - Inner view of the system.
    - Abstraction & Encapsulation are complementary to each other.
- Information hiding
    - Making members private in class so that they can be accessed outside the class only thorugh well-defined methods.
- Implementation

```
class Time {
    private int hour, minute, seconds;
    // methods
}
```

What are advantages of Object Oriented Programming?

- Data security

```
struct Time {
    int hour, minute, seconds;
};
// ....
struct Time t1;
t1.seconds = 100; // no compiler error -- may cause business logic to fail
// ...
incTime(&t1);
```

```
class Time {
    private int hour, minute, seconds;
    // methods ... setters/getters
    public void setSeconds(int sec) {
        if(sec < 0 || sec >= 60)
            throw exception;
        seconds = sec;
    }
};
// ....
struct Time t1;
t1.setSeconds(100); // throw exception
```

```
// ...
t1.incTime();
```

- Modularity
    - Namespaces/Packages/Modules
    - Classes
- Reusability (Association & Inheritance)

What is class and object? Give real-life example.

- Class is blue-print of the object. Class has fields (data) and methods (operations).
- Object is an instance of the class. One class can have multiple objects.
- Class is a logical entity.
- Object is a "physical" entity. Allocates memory.

What are characteristics of object? Explain them.

- State -- Values of data members
- Behaviour -- Operations performed/Methods called on state
- Identity -- Uniqueness of the object (e.g. address)

What is the need of getter and setter functions in class?

- Controlled access to the private fields
    - Programmer can choose which fields need to provide getter, setter or both.
    - Programmer can add validation logic into setter methods.

What is polymorphism? What are its types? Explain them with examples. What is the difference between function overloading and function overriding?

- Polymorphism -- Taking many forms.

    - Same name -- Different imlementations
- Compile time/Static Polymorphism
    - Function with same name but different arguments in same scope -- Function overloading.
    - Internally compiler assigns different name to each method (Name mangling).
    - Compiler decides which function to be called (Early binding), depeneding on number/type of arguments.

```
int add(int a, int b) { // addii
    return a + b;
}
int add(int a, int b, int c) { // addiii
    return a + b + c;
}
```

```
res = add(11, 22);
res = add(11, 22, 33);
```

    - False polymorphism
- Run time/Dynamic Polymorphism
    - Function with same name and same arguments redefined in derived class -- Function overriding.
    - Internally uses method table/virtual function table -- dynamic method dispatch.
    - Function to be called is decided at runtime (Late binding), depeneding on type of object (not the reference).

```
class Base {
    public void print() {
        System.out.println("Base.print() called");
    }
}
class Derived extends Base {
    public void print() {
```

```
        //super.print(); // Base::print();
        System.out.println("Derived.print() called");
    }
}
```

```
Base obj1 = new Base();
obj1.print(); // call Base.print()
Base obj2 = new Derived();
obj2.print(); // call Derived.print()
```

- True polymorphism

What is function overloading? Which are the rules of function overloading? Why return type is not considered in function overloading?

- Function with same name but "different" arguments in same scope -- Function overloading.
- Arguments should differ in count, data type, and/or order.
- Return type is not considered, because collecting return value is not compulsory. Compiler will not be able to resolve the function to be called.

```
int divide(int a, int b) {
    return a / b;
}
double divide(int a, int b) {
    return (double)a / b;
}
```

```
int r1 = divide(15, 5);
double r2 = divide(15, 5);
```

```
divide(15, 5);
```

What are different types of hierarchy? When to use which one?

- is-a hierarchy -- Inheritance (kind of)
  - Student is a person.
  - ~~Student has a person.~~
- has-a hierarchy -- Association (part of)
  - Emloyee has a Joining date.
  - ~~Emloyee is a Joining date.~~
- use-a hierarchy -- Dependency
- creates-a hierarchy -- Instantiation

Why constructor is considered special member function of the class?

- Constructor has same name as of class name.
- Constructor doesn't have any explicit return type.
- Constructor is automatically called when object is created.

What is object slicing? Explain object slicing in context of upcasting?

- Assigning derived class object address to base class reference is called as "Up-casting".
- When derived class object is assigned to base class reference, through that base class reference one can access only the base class members. This is called "Object slicing".

```
Person p = new Employee();
System.out.println(p.getName());
System.out.println(p.getSalary()); // not allowed -- compiler error
```

What is down-casting and when it is required?

- Assigning base class reference to derived class reference is called as "Down-casting".

```
void displayStudent(Person p) {
    System.out.println(p.getName());
    Student s = (Student)p; // Down-casting
    System.out.println(s.getMarks()); // allowed
}
```

```
displayStudent(new Student(...));
```

```
void displayEmployee(Person p) {
    System.out.println(p.getName());
    Employee e = (Employee)p; // Down-casting
    System.out.println(e.getSalary()); // allowed
}
```

```
displayEmployee(new Employee(...));
```

```
void display(Person p) {
   System.out.println(p.getName());
   if(p instanceof Emloyee) {
       Employee e = (Employee)p; // Down-casting
       System.out.println(e.getSalary()); // allowed
```

```
    }
    if(p instanceof Student) {
        Student s = (Student)p; // Down-casting
        System.out.println(s.getMarks()); // allowed
    }
}
```

```
display(new Person(...));
display(new Student(...));
display(new Employee(...));
```

What do you know about association, composition and aggregation. Explain with the help of example.

- has-a relationship -- Association
- Composition (Tight coupling) -- part of

```
class engine {
    // ...
};
class car {
    engine e; // tight coupling
    // ...
};
```

- Aggregation (Loose coupling)

```
class driver {
    // ...
```

```
};
class car {
    driver *d; // loose coupling
    // ...
};
```

What are different types of inheritance? Explain with the help of example. What are problems with multiple inheritance?

- Inheritance represents: is-a relation.
- Generalization to Specialization.
- Person --> Emloyee --> Manager --> HRManager
- Person --> Student
- Types
    - Single inheritance
    - Multiple inheritance
    - Multi-level inheritance
    - Hierarchial inheritance
    - Hybrid inheritance
- Problems in multiple inheritance: Ambiguity

```
class bird {
public:
    void fly() {
        // ...
    }
};
class aeroplane {
public:
    void fly() {
        // ...
    }
```

```
};
class birdplane: public bird, public aeroplane {
};
```

```
birdplane b;
b.fly(); // Ambiguity: bird::fly() or aeroplane::fly() --> Compiler Error
    // b.bird::fly(); --> bird::fly()
    // b.aeroplane::fly(); --> aeroplane::fly()
```

Which are the different types of design pattern? Explain singleton design pattern.

- Design pattern is standard solutions to well-known problems in software development/coding.

1. Creational: These patterns are designed for class instantiation. They can be either class-creation patterns or object-creational patterns. e.g. Factory Method, Abstract Factory, Builder, Singleton, Object Pool, and Prototype.
2. Structural: These patterns are designed with regard to a class's structure and composition. The main goal of most of these patterns is to increase the functionality of the class(es) involved, without changing much of its composition. e.g. Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Private Class Data, and Proxy.
3. Behavioral: These patterns are designed depending on how one class communicates with others. e.g. Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Null Object, Observer, State, Strategy, Template method, Visitor.

- Singleton class in Java: Have single object in whole appln.

```
class MyClass {
    // fields & methods

    private MyClass() {
        // ctor is private, so cannot create object outside the class
    }
```

```
    private static MyClass obj;
    static {
        // object is created only once -- while class loading
        obj = new MyClass();
    }

    // return same obj each time method is called
    public static MyClass getObj() {
        return obj;
    }
}
```

```
class MyClass {
    // fields & methods
    private:
        MyClass() {
            // ctor is private, so cannot create object outside the class
        }
        static MyClass *obj;

    // return same obj each time method is called
    public:
        static MyClass* getObj() {
            if(obj == NULL)
                obj = new MyClass;
            return obj;
        }
};

MyClass* MyClass::obj = NULL;
```