JAVA

→ First letter of class name should always be in uppercase.

→ syntax for declaring a string variable:-

```
String variable_name;
```

→ syntax for declaring an object :-

```
Class_name variable_name = new Class_name();
```

new → Keyword, Class_name() → Constructor

→ constructor name is the same as of class name and it doesn't have return type just like in C++.

→ syntax for main method in java:-

```
public static void main (String arg[])
```

public, static → Keywords

→ There are 3 access specifiers public, private and protected just like in C++.

→ main is declared publicly so that it can be accessed from outside.

→ static implies there are no objects to that class/method.

→ syntax for printing in output screen:-

```
System.out.println ("Faculty name:- " + fac_name);
```

+ → concatenation operator

```
class Faculty
{  Int fac-id;
   string fac-name;
   void display()
   { System.out.println("Faculty ID:-" + fac
     System.out.println("Faculty name:-" + fac
   }
}
public static void main(string args[])
{   Faculty a= new Faculty();
    a.fac-id = 1;
    a.fac-name = "Amit";
   Faculty b= new Faculty();
    b.fac-id = 2;
    b.fac-name = "Sumit";
    a.display();
    b.display();
}
```

→ Java complete reference

→ Object oriented programming – Ali Behrami

: Write a java program to find if one number is greater than the other.

```
class Myclass
{  int x, y;
   public static void main(string args[])
   {  Myclass a = new Myclass()
      a.x= 10; a.y=15;
```

```
if(a.x < ba.y)
System.out.println("The greater one is "+ a.y);
else
System.out.println("The greater one is "+ a.x);
}
```

} → Relationship between two objects through member functions / methods is known as aggregation.

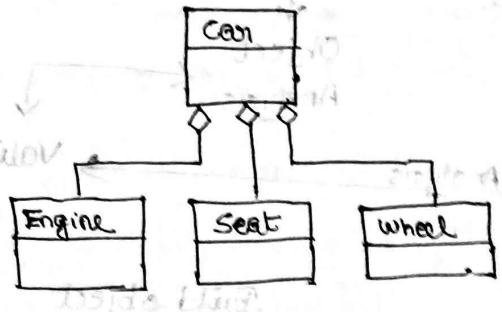
→ Association is used to establish relationship between two objects of different classes.

→ Cardinality specifies the number of instances of one class. may relate to a single instance of an associated class.

→ When a book in a library is issued to only one user it is known as unidirectional associativity.

→ When multiple books are issued to multiple users it is known as bidirectional associativity.

→ A client requesting a service is an unidirectional associativity.



— Procedure to draw class diagram.

→ When same method name and parameteres are used in different classes, it is known as method overriding.

Eg:-
```
class Birds extends Animals
{ void fly (string destination)
  { location = destination; }};
```

→ When the same method name is used in different classes but with different parameters, it is known as method overloading.

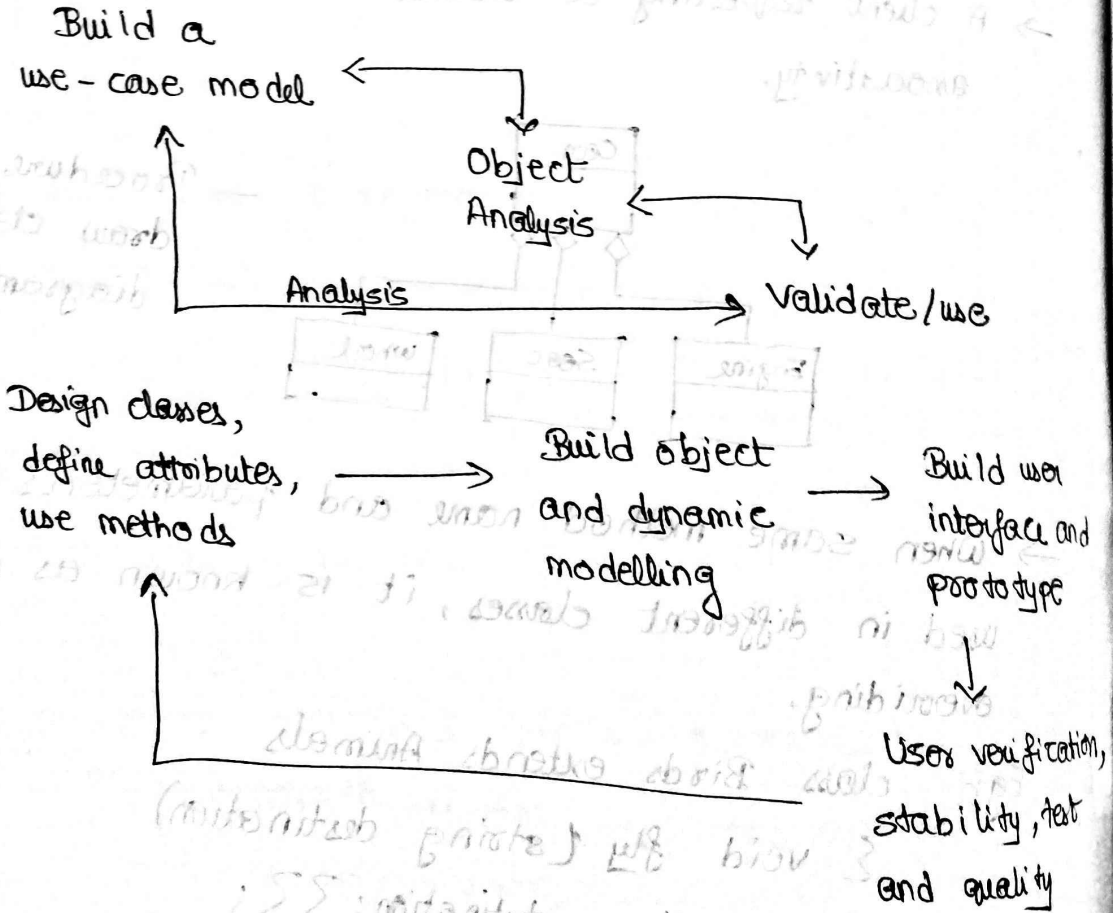
→ Super. method-name is also used to overide meth.

→ Java has inbuilt default constructor which sets all values to zero's.

→ Person (string name, int age)
{ this.name = name;
this.age = age; }

if the variable name in parameter and the variable inside the class are same, "this" operator is used to denote the variable inside the class.

Object Oriented Software Development Life Cycle



this operator

→ this keyword is used to refer to the current class

→ this() function is used to call the current class constructor.

GETTING INPUT VIA CONSOLE

```
Scanner variable_name = new Scanner(System.in);
   int a;
    a = variable_name.nextInt();
     float b;
     b = variable_name.nextFloat();
      string c)
     c = variable_name.nextLine();
```

→ The above syntax is followed for getting input from console for integer, floating point and string variables respectively.

CONVERTING STRING TO INTEGER/ OPERATIONS ON LARGE NUMBERS

```
String number1, number2);
Scanner.in = new Scanner(System.in))
 number1 = in.nextLine();
 number2 = in.nextLine())
BigInteger first = new BigInteger(number1);
```

BigInteger ← data type equivalent of long in C

```
BigInteger second = new BigInteger(number2);
Big Integer sum;
 sum = first.add(second);
```

→ java.util.Scanner is imported if the user wishes to use Scanner class.

Eg:- import java.util.Scanner;

→ math.BigInteger is imported if the user wishes to use BigInteger class.

Eg:- import java.~~*~~math.BigInteger;

→ java.util.* is imported if the user wishes to use the multiplication operator (*).

STRING METHODS

→ var_name.length() function returns the length of the string variable var_name.

→ var1 = var2.replace("a","b");

The above statement replaces all 'a' in var2 to 'b' and stores the new string in var1,

→ var1 = var2.concat("abc");

The above statement concatenates "abc" to var2 and stores it in var1. where var1 and var2 are string variables.

BOOLEAN DATA TYPE

→ Syntax:-

```
boolean a = true;
```

(true → keywords.)

→ Static Initialisation:-

```
int a = 5;
int b = a;
```

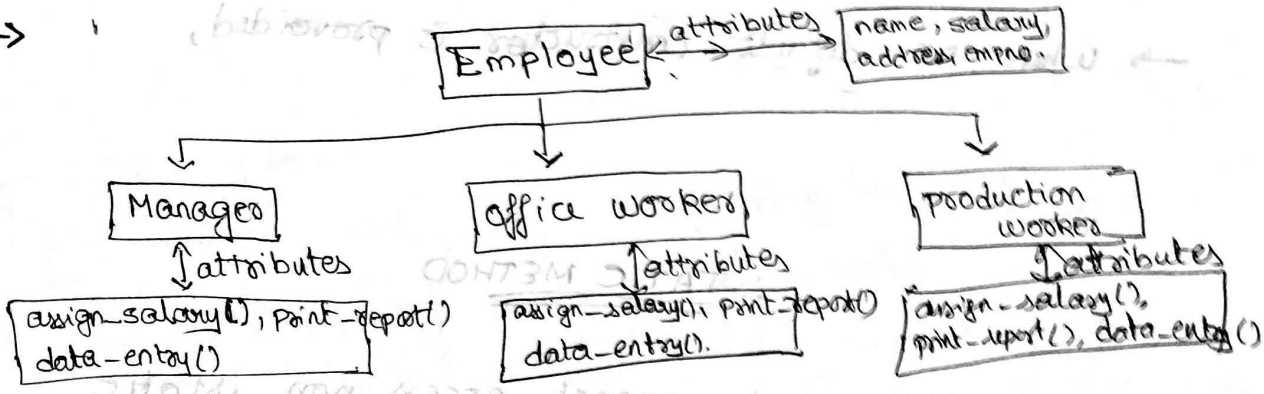
→ Dynamic initialisation:-

```
Scanner input = new Scanner (System.in);
int a = input.nextInt();
int i, sum = 0;
for (i=0; i<a; i++)
    sum += input.nextInt();
```

→ Array Declaration:-

```
int arr[] = new int [108];
int arr[] = {1,2,3};
```

→



→ Some methods may be repeated across inherited class albiet with a modification.

→ Common attributes are declared under the parent class employee.

→ Unique attributes are declared under children classes.

Static Variable

→ To count the number of students, we can declare a static variable in class college and increment it in constructor, so that the value of counter increases with the addition of every new student.

→ But if that static variable's value is to be displayed, 1 will be displayed everytime irrespective of the actual value of static variable because that static variable is a class member. (citation needed)

CONSTRUCTORS

→ There are two types of constructors in Java namely default constructor and parameterised constructor

→ When no default constructor is provided,

STATIC METHOD

→ Static methods cannot access non static members of a class.

→ super keyword is used to access members of parent class unlike this keyword which is used to access members of current class.

→ super and this operators cannot be used in static methods.

Eg:-

```
class C1
{ int i; // non static variable
  static String name = "CSE"; // static variable
 static update() // static method
  { name = "Computer Science"; }
  void display() // non static method
  { System.out.println(name); }.
 public static void main(String args[])
  { update();
     display(); // compile time error
  }
}
```

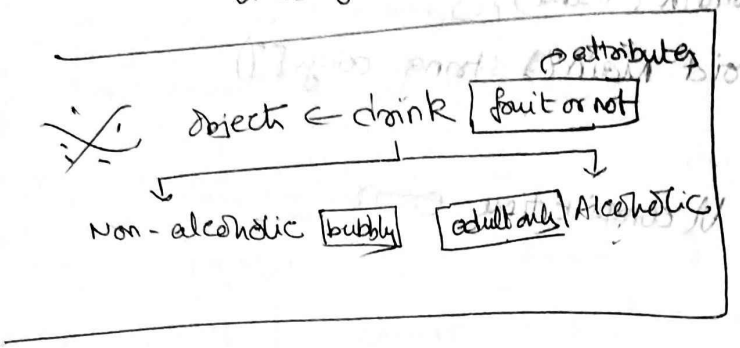
→ display() member function cannot be called without a class object because it is not a static method.

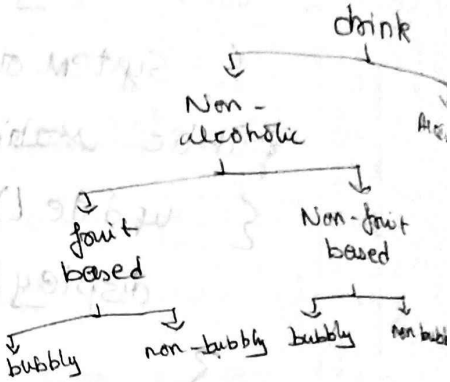
METHOD OVERLOADING

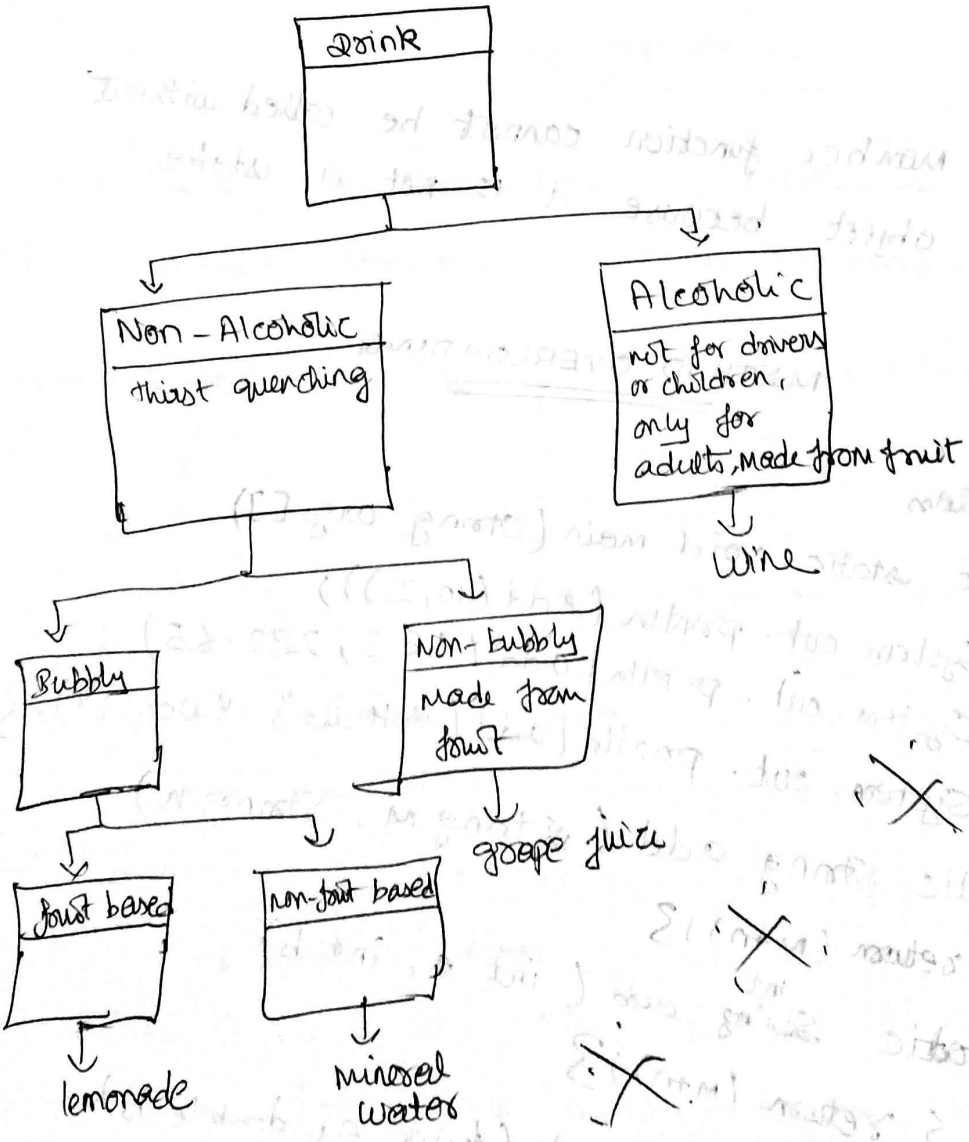
Eg:

```
class Myclass
{ public static void main(String args[])
  { System.out.println(add(10,35))
    System.out.println(add(45.3, 282.65));
    System.out.println(add("Hello", "World"); }
 Static String add(String M, String n)
  { return(M+n); }
 static int add(int a, int b)
   { return(M+n); }
 Stesting double add(double a, double b)
   { return(a+b); }
```

2) Alcoholic are'nt for drivers / children
Non-Alcoholic drinks are thirst quench
Wine is made of grape and is for adults only
Grape juice is made from grape and tastes like fr
Mineral water is bubbly and doesn't taste like fruit
Lemonade is bubbly and tastes like fruit.







Validation, verification.

→ Correspondence ensures that the program meets the user requirements or not

→ Validation is the task of practicing correspondance.

→ Correctness measures the consistancy of product. requirement unit design specification.

→ Verification is the excercise of determining correctness.

→ Validation is the first stage, verification is the second stage

→ Fig. 3.4, 3.5

→ Design a large number of simple classes rather than small number of complex classes

→ Java takes care of deleting dynamically allocated memory after its use automatically unlike C++ where we have to manually delete it using delete() function. This is known as garbage collection.

→ final keyword is used to call the parent class constructor.

→ finalize() method is similar to destructor. It provoides the last minute operations on the file / variable before it gets automatically deallocated. It should be defined in protected mode inside a class.

STACK CLASS

```
class Stack
{  int stck[] = new int[10];
   int top;
   Stack()
   { top = -1; }

   void push(int ele)
   {  if(top == 9)
      System.out.println("stackFullException");
      else
      stck[++top] = ele; }

   int pop()
   {  if(top<0)
      { System.out.println("stackEmptyException");
        return 0; }
        else
        return(stck[top--]); }
};
```

→ Generalisation ▲ (symbol)

→ Specialisation ▼ (symbol).

→ Generalisation enables inheritence of attributes and operations. (Its' a bottom-up approach).

→ Specialisation is a top down approach.

→ Aggregation relates parts to whole. (Eg:- a door is a part of car).

→ Association is a miscellaneous relationship between classes. (Eg:- Patient having a doctor's appointment).

2. Identify for the following

i. Mail has address.

Aggregation.

ii. Mail uses address for delivery.

Association

iii. Customer has address

Aggregation

iv. customer resides at address.

Association

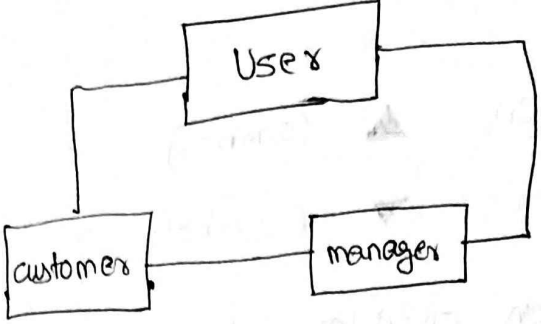
v. TV includes screen.

Aggregation

vi. TV sends information to the screen.

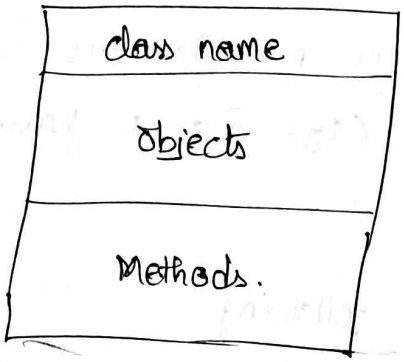
Association.

vii. Customer is a user Generalisation / Specialisation



→ The above diagram is an example of unary association.

→ Diagram syntax :-



→ Multiplicity is the number of inherited classes of a particular class.

→ If multiplicity is exactly one :- 1

→ " zero or more :- 0...*

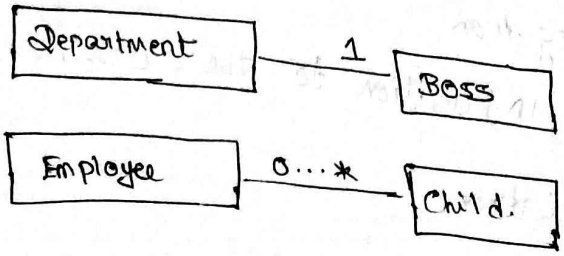
→ " 1 or more :- 1...*

→ " zero or 1 :- 0...1

→ " 2 to 4 :- 2...4

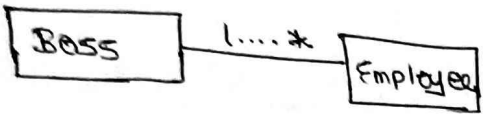
→ " Multiple disjoint ranges :- 1...5,7 (1-5 or 7).

Eg :-

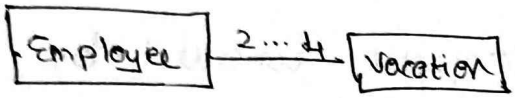


A department has exactly one boss.

An employee can have 0 or more children



A boss can handle 1 or more employees.



An employee can take 2-4 vacations.

→ Qualifier is an association attribute.

→ Eg:- A person object maybe associated to a bank object.

→ An attribute of this association is the account number.

→ Account number is the Qualifier of this (bank-person) association.

INHERITANCE

→ Arrow mark in class diagram should be towards the parent class.

→ class B extends A { } means B inherits A.

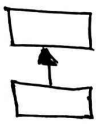
→ Code reusability is the usage of the same method code in the child class.

→ Code sharing is the sharing of code from parent class to child class.

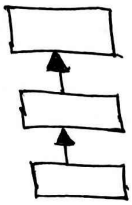
→ There are 5 types of inheritance namely single level, multi-level, multiple, hybrid and heirarchial.

→ Java doesn't support multiple inheritance.

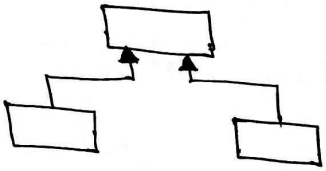
→ single level :-



→ Multi-level :-



→ Heirarchial :-



→ As java doesn't support multiple inheritance, it also doesn't support hybrid inheritance.

→ final keyword is used for initialising constant variables. (ie. it's value cannot be changed in the program).

Eg:- final int a = 10; // like const keyword in C.

→ finalize() method is called before garbage collection to print / perform operations before the object is destructed. (like destructor function in c++).

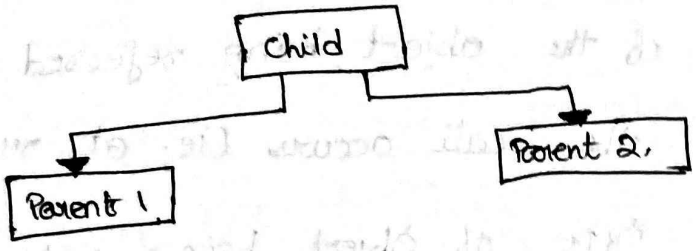
Eg:- for Heirarchial inheritance :-

```
class parent {}
class child1 extends parent {}
class child2 extends parent {}
```

→ The parent class constructor would be called first if a child class variable is called just like in c++.

→ Inside the child class's constructor super() method must be called which in turn will call the parent class's constructor.

→ Multiple inheritance :- (NOT SUPPORTED IN JAVA).



→ If super.fn() is called fn() method of it's direct parent class will be called.

→ super keyword is has similar features as of this operators.

→ super operator can be concatenated.. Super operator is used when the parent and the child class have a same name (method overwriting).

DYNAMIC METHOD DISPATCH

→ The problem of which method is to be chosen in case of method overwriting will be decided during runtime instead of compile time.

→ It is a mechanism by which a call to an overwritten method is resolved at runtime rather than during compile-time.

→ When an overridden function is called through a superclass reference, java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. (ie. at runtime).

→ It is the type of object being referred to and not the type of reference variable that is important.

```
class A {
    void callme() {
        print("Inside A"); }}
class B extends A {
    void callme() {
        print("Inside B"); }}
class C extends A {
    Void callme() {
        print("Inside C"); }}
class Dispatch {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
    A r ; // obtain a reference of type 'A'.
        r = a; // r refers to object 'a'.
        r.callme();
        r = b;
        r.callme();
        r = c;
        r.callme(); }}
```

→ A pointer of a type can also refer to its child class type in Java. CITATION NEEDED.

U

PACKAGE

→ Package is a place where classes are stored
→ Directory name must match the package name exactly
→ More than one file can include the same package statement.
→ We can also create a hierarchy of packages.
→

∴

Limitation of Interfaces:-

→ It is useful only when it needs to be implemented multiple number of times.
→ By default all interfaces are public.
→ An use case is a special flow of events through the system.
→ An actor plays the role with respect to system.
→ The actors communicates with the system's use cases.
→ Activity diagram is a variation of the state diagram in which the states are activity reflecting the performance of activity and the transitions are treated as a completion of activity.

→ State is a set of values that describes the object at a specific point in time.