**Definitions CFGs**

Class Definiton CFG

**Note:-**

**Return statement SST me nhi rkhana or return st optional h**

<class\_def> → <class\_header> class ID <extend\_st> <implement\_st> : ( <class\_body> )

**Agar AM kch bhi nhi h to bydefault public hoga**

<class\_header> 🡪 AM <ch2> | final | €

<ch2> 🡪 final | €

<extend\_st> → extends ID |  €

<implement\_st> → implements ID <interface\_rec> |  €

<interface\_rec> → , ID <interface\_rec> |  €

<class\_body> → AM <cb3>

<class\_body> → static <amh3> <cb2>

<class\_body> → final <cb2>

<class\_body> → <cb2>

<class\_body> → €

<cb2> 🡪 DT <cb2a>

<cb2> 🡪 String <cb2a>

<cb2> 🡪 dict <cb2b>

<cb2> 🡪 ID <cb2c>

<cb2> 🡪 void ID { <params\_list> } : <body> <class\_body>

<cb2a> 🡪 ID <cb2a1>

<cb2a> 🡪 [ ] <arr\_mul> ID { <params\_list> } : <body> <class\_body>

<cb2a1> 🡪 <dec1> <class\_body> | { <params\_list> } : <body> <class\_body>

<cb2b> 🡪 ID <cb2b1>

<cb2b> 🡪 [ ] <arr\_mul> ID { <params\_list> } : <body> <class\_body>

<cb2b1> 🡪 <init\_dict> <list\_dict> <class\_body> | { <params\_list> } : <body> <class\_body>

<cb2c> 🡪 ID <cb2c1>

<cb2c> 🡪 [ ] <arr\_mul> ID { <params\_list> } : <body> <class\_body>

<cb2c1> 🡪 <init> <list> <class\_body> | { <params\_list> } : <body> <class\_body>

<cb3> 🡪 DT <cb2a>

<cb3> 🡪 String <cb2a>

<cb3> 🡪 dict <cb2b>

<cb3> 🡪 ID <cb3a>

<cb3> 🡪 void ID { <params\_list> } : <body> <class\_body>

<cb3> 🡪 static <amh2a> <cb2>

<cb3> 🡪 final <cb2>

<cb3a> 🡪 <cb2c> | { <params\_list> } : ( <constr\_body> ) <class\_body>

<amh2> 🡪 € | static <amh2a> | final

<amh2a> 🡪 final | €

<amh3> 🡪 final | €

Ye hatani hain ( confirm krden

<attrb> → <attrb&method\_header> <dec>

<dec> 🡪 DT ID <dec1> | String ID <dec1> | <dec\_dict> | <dec\_obj\_or\_enum>

<dec1> 🡪 <init\_var> <list\_var> | [ <OE> ] <arr\_size> <init \_arr> <list\_arr>

Yaha tk)

<dec\_obj\_or\_enum> → ID ID <init> <list>

<init> 🡪 = <init2>

<init> 🡪 €

<init2> 🡪 new ID { <args-list> } <list\_obj> | <OE>

<list> 🡪 , ID <list2>

<list> 🡪 €

<list2> 🡪 <init> <list>

<method> → <attrb&method\_header> <func\_def>

<constr> → AM ID { <params\_list> } : ( <constr\_body> )

<constr\_body> →  <pointer\_st> <MST>

<pointer\_st> → TS { <args\_list> } <pointer\_st> |  €

<args\_list> <OE> <list\_args> | ∈

<list\_args> , <OE> <list\_args> | ∈

**Interface CFG**

**Note:-**

**Interface Defintion CFG syntax java based:**

interface InterfaceName {

// Constant variables (optional)

int CONSTANT\_VARIABLE = 100; // By default, variables in interfaces are public, static, and final

// Abstract methods (method declaration without implementation)

void method1();

int method2(String param);  
  
syntax 2:  
  
interface Calculator {

// Method with parameters

int add(int a, int b);

int subtract(int x, int y);

}  
  
hmari language syntax:   
public interface Calculator extends plus , minus , multiply : (  
int var = 100

int add{int a, int b}

int subtract{int x, int y}

void print{}

)

**CFG:**

<interface\_def> → <interface\_header> interface ID <extend\_st\_interface> : ( <interface\_body> )

<interface\_header> 🡪 AM | ∈  
  
<extend\_st\_interface> --> extends ID <interface\_rec> | ∈  
  
<interface\_rec> --> , ID <interface\_rec> | ∈

<interface\_body> → <attrb><interface\_body> | <method\_interface> <interface\_body>

<attrb> 🡪 <attr\_header> <dec>   
  
<attrb\_header> 🡪 AM static final | ∈

<dec> --> <dec\_var> | <dec\_arr> | <dec\_obj> | <dec\_enum> | <dec\_dict>

<method\_interface> --> <interface\_method\_header> <func\_def\_interface>

Methods of our interface will never be static

<interface\_method\_header> --> AM | ∈

<func\_def\_interface>--> DT ID { <params\_list> }  
<func\_def\_interface>--> DT [ ] <arr\_mul> ID { <params\_list> }

<func\_def\_interface>-->String ID { <params\_list> }

<func\_def\_interface>--> String [ ] <arr\_mul> ID { <params\_list> }

<func\_def\_interface>--> ID ID { <params\_list> }

<func\_def\_interface>--> ID [ ] <arr\_mul> ID { <params\_list> }

<func\_def\_interface>--> dict ID { <params\_list> }  
<func\_def\_interface>--> dict [ ] <arr\_mul> ID { <params\_list> }

**Func def**

Syntax

Returntype ID { int a,char b, bool c }: (

SST

…

)

<func\_def> 🡪 DT <fnd2>

<func\_def> 🡪 String <fnd2>

<func\_def> 🡪 void ID { <params\_list> } : <body>

<func\_def> 🡪 ID <fnd2>

<func\_def> 🡪 dict <fnd2>

<fnd2> 🡪 ID { <params\_list> } : <body>

<fnd2> 🡪 [ ] <arr\_mul> ID { <params\_list> } : <body>

<params\_list> 🡪 <params> | €

<params> → DT <params2>

<params> → ID <params2>

<params> → String <params2>

<params> 🡪 dict <params2>

<params2> 🡪 ID <list> | [ ] <arr\_mul> ID <list>

<list> → , <params> | €

<arr\_mul> 🡪 [ ] <arr\_mul> | €

<body> → ( <MST> <return\_st> )

<SST> → <while\_st> | <dec\_st> | <if\_st> | …

<MST> <SST>  <MST> | €

<return\_st> → € | return <OE>

**Enum Def**

Syntaxes:

enum Direction : (

North,

South,

East,

West

)  
  
enum Direction : (

North = 11,

South =33,

East =56,

West=89

)

CFG

<enum\_def> --> Enum ID : ( ID <list> )

<list> --> , ID <init\_enum\_def> <list> | €

<init\_enum\_def> --> = <OE> <list> | €