# OBJECT ORIENTED PROGRAMMING

VIVEK DUTTA MISHRA



VIVEK@CONCEPTARCHITECT.IN

# QUIZ

Q1. WHEN WAS C++ CREATED?

- 1979 -1983

Q2. WHAT WAS THE ORIGINAL NAME OF C++?

- C WITH OBJECTS
- C WITH CLASSES
- D
- TURBO C

Other names considered Included

- New C
- ++ C

# QUIZ

Q3. ORIGINALLY C++ HAD NO COMPILER OR INTERPRETER – HOW WAS A C++ CODE EXPECTED TO EXECUTE?

A. THEY CREATED A TRANSPILER TO TRANSLATE C++ CODE TO C CODE.

Q4. WHY NOT A COMPILER OR INTERPRETER?

- COMPILER/INTERPRETER ARE MACHINE DEPENDENT
  - DEVELOPMENT TAKES MORE TIME
- TRUE COMPILER SHALL BE CREATED LATER
  - QUICK DISTRIBUTION

Q5. IS IT POSSIBLE TO TRANSLATE C++ TO C?

A. LETS SEE.

```
struct
class  Triangle{
  private:
     int s1,s2,s3;
  public:
    void set(int x,int y, int z){
    s1=x;
    s2=y;
    s3=z;
    }
    int perimeter(){
          return s1+s2+s3;
    }
};


void main(){
     Triangle t1,t2;
     t1.set(3,4,5);
     int p= t1.perimeter();
}
```

C doesn't have class keyword. But we can use struct instead

C Doesn't have scope or any ... be

C doesn't support functions within class
So the function must be out of struct.

...ks for ...emoves the scope from the source code

```
struct Triangle{
   int s1,s2,s3;
};

int Triangle_set(int x,int y,int z){
        s1=x;
        s2=y;
        s3=z
}
int perimeter()
{
        return s1+s2+s3;
}

void main()
{
        Triangle t;
        t.set(3,4,5);
        int p=t.perimeter();
}
```

Q. How to differentiate between the set and perimeter of a Triangle with the set and perimeter of a Circle???

A. We can prefix class name to function name

```
struct Triangle{
   int s1,s2,s3;
};

int Triangle_set(Triangle * this, int x,int y,int z){
      this-> s1=x;
      this-> s2=y;
      this-> s3=z
}

int Triangle_perimeter(Triangle * this )
{
      return this-> s1+ this-> s2+ this-> s3;
}

void main()
{
      Triangle t1,t2 ;
      Triangle_set(&t1,3,4,5); // t1.set(3,4,5)
      Triangle_perimeter(& t );
}
```

Q. How Do I Know which Triangle t1 or t2?
(t is missing in transpilation)

**The Birth of**

# ‘this’

**keyword**

Let pass a new Parameter to specify the object to be used with set /perimeter

# THE REAL CHALLENGE – OVERLOADED FUNCTION

```
int divide(int x,int y){
        return x/y;
}

double divide(double x,double y){
      return x/y;
}

void main()
{
        int r1= divide(7,2) ; //3
        double r2= divide(7.0,2.0); //3.5
}
```

Q. C Doesn't support overloading.

We can Change the name of functions based on parameters Passed.

Known as

- Name Decoration or
- Name Mangling

# WHY ARE WE DISCUSSING ALL THIS!!!

- C++ IS A SOFTWARE PRODUCT
- ITS DESIGN IS GUIDED BY CERTAIN DESIGN DECISIONS
- THOSE DECISION INFLUENCED CERTAIN LONG TERM GOALS
- TWO KEY DECISION
  1. C++ EVOLVED AS C WITH CLASSES → NEW C → ++ C → C ++
  2. A TRANSPILER AS A STOPGAP SOLUTION

# C++ IS A BETTER C

- C++ WAS DESIGNED AS BETTER C NOT AS A NEW LANGUAGE
- C++ WAS DESIGNED AS BACKWARD COMPATIBLE WITH C.
  - EVERY VALID C PROGRAM WILL ALSO BE A VALID C++ PROGRAM
- IS THIS A PROBLEM?
  - C++ DESIGNER HAD NO OPTION TO REMOVE ANY FEATURE WHICH DEEMED ERROR PRONE OR UN-NECESSARY
  - OOP IS OPTIONAL
    - YOU MAY CHOOSE TO WRITE AS MUCH OR AS LESS AS YOU WANT

Most of the time we chose not to write a complete OO design !!!

# C++ TRANSPILER DESIGN

- C++ CODE SHALL BE TRANSLATED TO C AND THEN COMPILED USING C COMPILER
- **IS THIS A PROBLEM?**
- EVERY C++ FEATURE NEED TO BE PERCEIVED AS TRANSPILABLE TO C
- SEMANTICAL COMPROMISES/CONSTRUCTS INTRODUCED TO ENABLE TRANSPILATION
  - THIS POINTER
  - NAME MANGLING
  - NO RUNTIME SCOPE

# CONSIDER THE BELOW CODE

```
class Shape{
  public:
   virtual int area(){
     return 0;
   }
};

class Line : public Shape{
  private:
   int area(){
     return -1;
   }
};

void main() {
   Line * line=new Line();
   line -> area();
   Shape * shape= line;
   shape->area();
}
```

**Line** has no area so area() is overridden in **private.**

**No Runtime Scope**

Not Permitted. The Scope is **private.**
**(Works as expected)**

area() is public in shape and is called. Due to virtual it wll end up calling **private** area from Line

# SUMMARY

- C++ DESIGNER WERE UNABLE TO REMOVE ANY UNDESIRABLE/UN-NECESSARY FEATURES
  - THIS IS THE COST TO BACKWARD COMPATIBILITY
- TRANSPILER WAS A STOPGAP SOLUTION BUT IT HAD PERMANENT IMPACT ON LANGUAGE DESIGN
  - NAME MANGLING MADE RUNTIME NAME IDENTIFICATION INFEASIBLE.
    - OVERLOADING IS A HACK, NOT A FEATURE
    - NO DYNAMIC PROGRAMMING MODEL (REFLECTION)
  - NO DYNAMIC RUNTIME SCOPE

# TAKE AWAY

- BACKWARD COMPATIBILITY MAY LIMIT YOUR DESIGN SCOPE
  - BACKWARD COMPATIBILITY CANT BE IGNORED
  - WE MUST CONSIDER HOW MUCH?
- SHORT-TERM GOAL SHOULD NOT OVERSHADOW LONG TERM DESIGN DECISION.

# FEW YEARS LATER…

- 1991-1992
- SUN-MICROSYSTEM SET DOWN TO CREATE A OS FOR SET-TOP BOX
- THEY WANTED TO DO IT USING OO APPROACH
- C++ WAS TOO LIMITING TO THEIR VISION
- THEY CREATED OAK
- OAK LATER EVOLVED INTO JAVA

# MOTIVATION OF JAVA

- C++ --
  - -- (SIMPLIFIED BY REMOVING UNWANTED FEATURES)
  - AN ORIGINAL DESIGN FROM SCRATCH
    - NO BACKWARD COMPATIBILITY (UNLIKE C++)
    - NOT BACKWARD COMPATIBLE TO EVEN ITS OWN PAST SELF
      - DEPRECATED.
  - GREAT ROOM FOR CLEAN DESIGN.

# WHAT GOT REMOVED

- C++ --
  - GLOBALS & PROCEDURAL PROGRAMMING
  - MACROS
  - TYPEDEF, UNION, STRUCT
  - MEMORY DEALLOCATION
  - DESTRUCTOR
  - FRIEND FUNCTION
  - MULTIPLE INHERITANCE
  - OPERATOR OVERLOADING
  - ENUM
  - TEMPLATES

Not all features removed were necessarily bad.

Java Re-introduced Enum and Template

Java may never introduced generics because developer had vocal critic on the subject.

Beaware!!!
Personal Egos are bad for your Design

# SUMMARY

- OUR DESIGN SHOULDN'T BE FOCUSED TOO MUCH ON BACKWARD COMPATIBILITY
  - IT SHOULD BE MORE FUTURE PROOF.
- SHORT-TERM GOAL MAY HAVE LONG TERM (TERRIBLE) CONSEQUENCES
- EGOS ARE BAD FOR OUR DESIGN.
  - BE OPEN.