class Icon

{

    float speed, glow, energy;

    int x, y;

    int subtype; // spinner, slider or hopper

    bool clockwise; // need for spinner

    bool expand; // need for spinner

    bool vertical; // need for slider

    int distance; // need for slider

    bool visible; // need for hopper

    int xcoord, ycoord; // need for hopper

    void spin() { }

    void slide() { }

    void hop() { }

    // constructor must set subtype: client must pass value

    public Icon(unsigned value)

    {

        subtype = value; // use enum for readability

        // and then use conditional to set associated fields

    }

    public void move()

    {

        if (subtype == 1) { spin(); }

        else if (subtype == 2)

        {

            slide();

        }

        else

        {

            hop();

        }

    }

// tedious subtype checking: subtype drives flair details

   public void flair()

   {

       if (subtype == 1) { spin(); }

       else if (subtype == 2)

       {

           slide();

       }

       else

       {

           hop();

       }

   }

}

code debt-

1. Only one class is doing all jobs acting as a **god class** violating **single responsibility principle**

2. Code is not easily adapted to change since presence of tightly coupled conditions.

3. No abstraction.

Public Class IconSelcetor:Icon

{

    float speed, glow, energy;

    int x, y;

Void move()

Void Flair()

}

Public Interface Icon

{

Void move()

Void flair()

}

Class spin :IconSelector{

    bool clockwise; // need for spinner

    bool expand; // need for spinner

    void spin() { }

void move(){}

void flair(){}

}

Class slide:IconSelctor{

    bool vertical; // need for slider

    int distance; // need for slider

void slide() { }

void move(){}

void flair(){}

}

Class hop:IconSelector{

    bool visible; // need for hopper

    int xcoord, ycoord; // need for hopper

    void hop() { }

void move(){}

void flair(){}

}