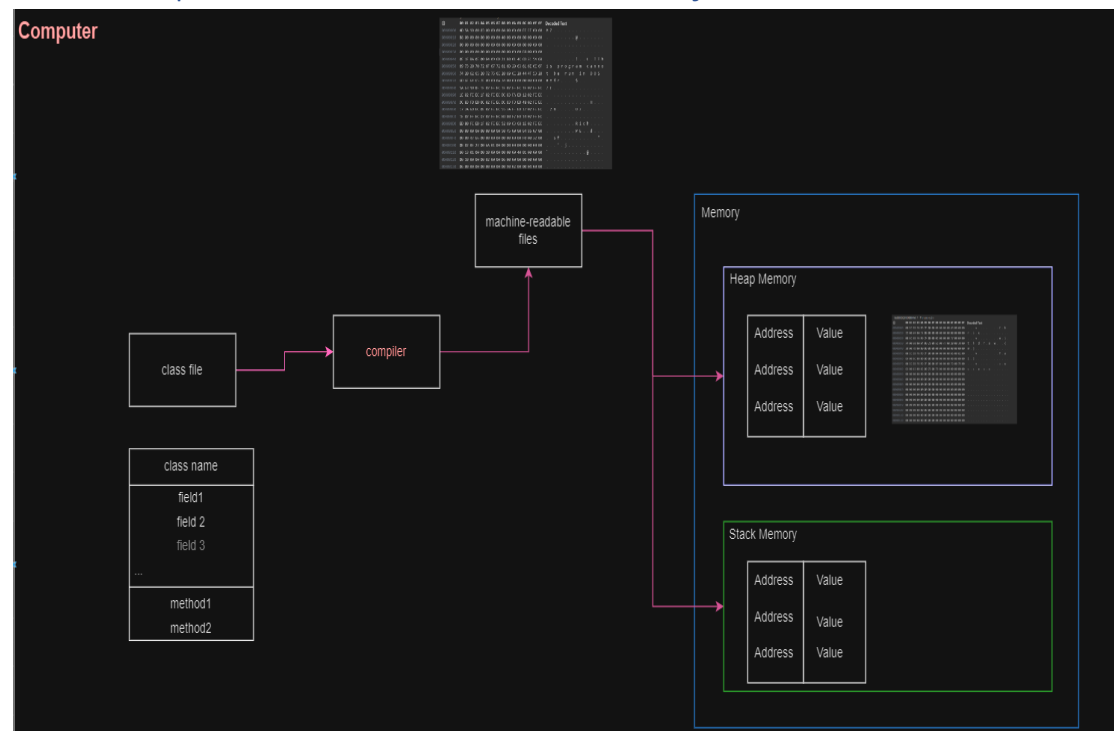


How computer react when create an object



(Figure 1)

Class & Object

As we can see from Figure 1, the essence of a class is a file that can be compiled into machine-readable code. And Object is a reference to an address allocated in memory.

We can create an object by a class file.

Constructor & method & filed & property

constructor

```
namespace RobotDodge
{
    3 references
    public class RobotDodge
    {
        7 references
        private Player _Player;
        6 references
        private Window _GameWindow;
        4 references
        private Robot _TestRobot;

        0 references
        public bool Quit
        {
            get { return _Player.Quit; }
        }

        1 reference
        public RobotDodge(Window gameWindow)
        {
            _GameWindow = gameWindow;
            _Player = new Player(gameWindow);
            _TestRobot = new Robot(gameWindow);
        }
    }
}
```

(Figure 2)

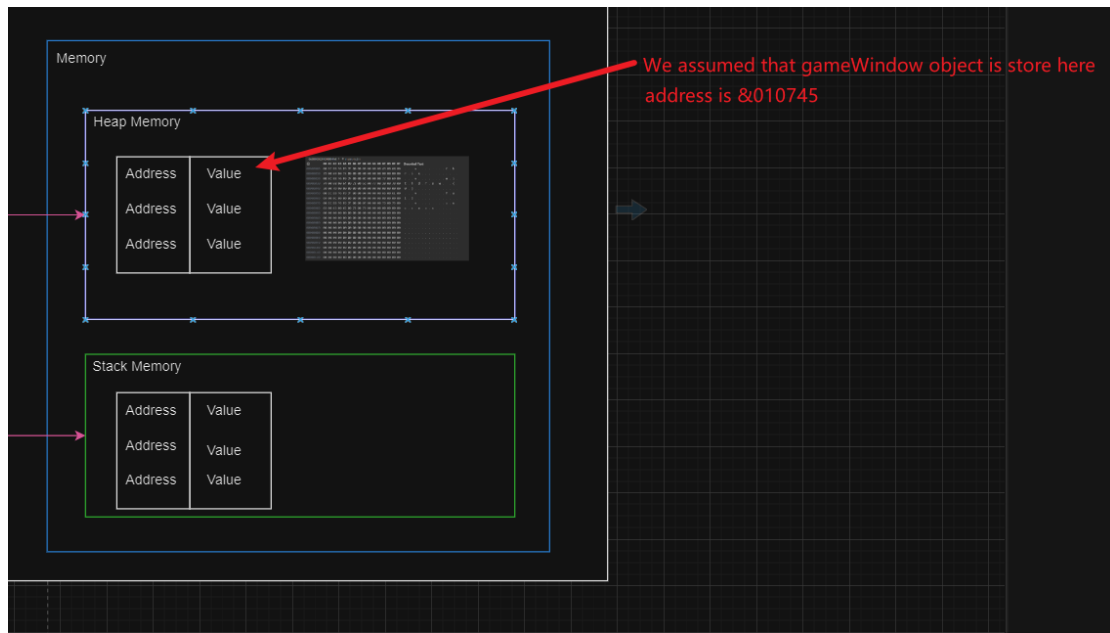
Constructor plays an important role in creating objects. When we want to assign certain properties to an object when it is initialized, we can pass the object through the constructor.

From Figure 2, a Window object **gameWindow** is passed to the constructor of **RoboDodge**, then **Robotdodge** object can be created with the filed **_GameWindow** whose reference is from **gameWindow**.

For better understanding:

Figure 3, we assumed that **gameWindow** object is stored in **&010745**;

Figure 4, **RoboDodge** object's field **_GameWindow** has been assigned the address **&010745**.



(Figure 3)

```
namespace RobotDodge
{
    3 references
    public class RobotDodge
    {
        7 references
        private Player __Player;
        6 references
        private Window __GameWindow; &010745
        4 references
        private Robot __TestRobot;

        0 references
        public bool Quit
        {
            get { return __Player.Quit; }
        }

        1 reference
        public RobotDodge(Window gameWindow) &010745
        {
            __GameWindow = gameWindow; &010745
            __Player = new Player(gameWindow);
            __TestRobot = new Robot(gameWindow);
        }

        1 reference
        public void HandleInput()
        {
            __Player.HandleInput();
            __Player.StayOnWindow(__GameWindow);
        }
    }
}
```

(Figure 4)

Method

Methods are used to describe the actions of an object and can correspond to the actions of objects in the real world.

When the program executes a method, the method will be loaded into the call stack, and the local variables created in the method will be recycled after the method call ends.

Methods accept two type of parameters, value types and reference types.

Figure 5: The value type passes a specific value

Figure 6: Reference types pass addresses

```
4 references
public void Move(double forward, double strafe)
{
    Vector2D movement = new Vector2D();
    Matrix2D rotation = SplashKit.RotationMatrix(0);

    movement.Y += forward;
    movement.X += strafe;

    movement = SplashKit.MatrixMultiply(rotation, movement);
    X += movement.X;
    Y += movement.Y;
}
```

(Figure 5)

```
2 references
public void StayOnWindow(Window limit)
{
    if (X < GAP)
    {
        X = GAP;
    }
    else if (Y < GAP)
    {
        Y = GAP;
    }
    else if (X + _PlayerBitmap.Width > limit.Width - GAP) {
        X = limit.Width - GAP - _PlayerBitmap.Width;
    }
    else if (Y + _PlayerBitmap.Height > limit.Height - GAP) {
        Y = limit.Height - GAP - _PlayerBitmap.Height;
    }
    else {
    }

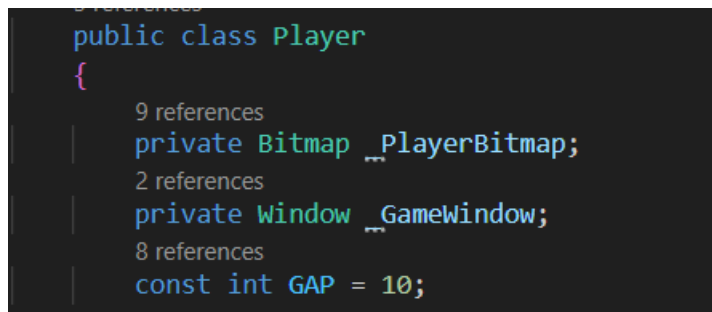
    Draw();
}
```

(Figure 6)

Field

Fields are used to store data for objects.

Figure 7: **_PlayerBitmap** and **_GameWindow** is **Player's** filed



```
public class Player
{
    9 references
    private Bitmap _PlayerBitmap;
    2 references
    private Window _GameWindow;
    8 references
    const int GAP = 10;
}
```

(Figure 7)

Property

Properties provide a controlled access to fields.

Figure 8: **X** is a read only property, which allows other objects get its value.

```

namespace RobotDodge
{
    3 references
    public class Player
    {
        9 references
        private Bitmap _PlayerBitmap;
        2 references
        private Window _GameWindow;
        8 references
        const int GAP = 10;
        8 references
        public double X
        {
            get;
            private set;
        }

        8 references
        public double Y
        {
            get;
            private set;
        }

        1 reference
        public bool Quit
        {
            get;
            private set;
        }
    }
}
1 reference

```

(Figure 8)

Control flow

Control flow help us to implement our logic.

Sequence

The program will execute the code step by step;


Figure 9: The program will be executed in order

```

1 reference
public void Draw()
{
    double leftX, rightX;
    double eyeY, mouthY;
    leftX = X + 12;
    rightX = X + 27;
    eyeY = Y + 10;
    mouthY = Y + 30;
    SplashKit.FillRectangle(Color.Gray, X, Y, 50, 50);
    SplashKit.FillRectangle(_MainColor, leftX, eyeY, 10, 10);
    SplashKit.FillRectangle(_MainColor, rightX, eyeY, 10, 10);
    SplashKit.FillRectangle(_MainColor, leftX, mouthY, 25, 10);
    SplashKit.FillRectangle(_MainColor, leftX + 2, mouthY + 2, 21, 6);
}

```

sequence



(Figure 9)

Selection

Selection is a logical filter that we can use to limit the execution of branches only when the condition I met; Usually, we use if to implement selection, but we can also use switch to implement selection.

```

1 reference
public void HandleInput()
{
    SplashKit.ProcessEvents();

    int vector = 5;
    if (SplashKit.KeyDown(KeyCode.UpKey) || SplashKit.KeyDown(KeyCode.WKey))
    {
        Move(-vector, 0);
    }
    else if (SplashKit.KeyDown(KeyCode.DownKey) || SplashKit.KeyDown(KeyCode.SKey))
    {
        Move(vector, 0);
    }
    else if (SplashKit.KeyDown(KeyCode.LeftKey) || SplashKit.KeyDown(KeyCode.AKey))
    {
        Move(0, -vector);
    }
    else if (SplashKit.KeyDown(KeyCode.RightKey) || SplashKit.KeyDown(KeyCode.DKey))
    {
        Move(0, vector);
    }
}

```

(Figure 10)

```

private void SetShipBitmap()
{
    switch (_kind)
    {
        case ShipType.Aquarii:
            _shipBitmap = SplashKit.BitmapNamed("Aquarii");
            break;
        case ShipType.Gliese:
            _shipBitmap = SplashKit.BitmapNamed("Gliese");
            break;
        case ShipType.Pegasi:
            _shipBitmap = SplashKit.BitmapNamed("Pegasi");
            break;
        default:
            _shipBitmap = SplashKit.BitmapNamed("Aquarii");
            break;
    }
}

```

(Figure 11)

Repetition

Loop is used to repeat a path a number of times and repeat code a variable a number of times.

Usually, there are two types of repetition, while and for. And the while loop also have two types, pre-test loops and post-test loops.

Figure 12: for loop can help you easily to repeat a path a number of times.

Figure 13: do while loop runs at least once.

Figure 14: while loops runs 0 to many times.

```

public void SumNumber()
{
    int sum = 0;
    for (int i = 1; i >= 10; i++)
    {
        sum += i;
    }
}

```

(Figure 12)


```
public static void Main()
{
    MenuOption userSelection;

    do
    {
        userSelection = ReadUserOption();
        Console.WriteLine(userSelection);
        switch (userSelection)
        {
            case MenuOption.TestName:
                TestName();
                break;
            case MenuOption.GuessThatNumber:
                RunGuessThatNumber();
                break;
            case MenuOption.Quit:
            default:
                break;
        }
    } while (userSelection != MenuOption.Quit);
}
```

(Figure 13)

```

private static void RunGuessThatNumber()
{
    int lowestVal = 1, highestVal = 100, guessVal = -1;
    int targetVal = new Random().Next(100) + 1;

    while (true)
    {
        Console.WriteLine("Guess a number between 1 and 100");
        guessVal = ReadGuess(lowestVal, highestVal);

        if (guessVal < targetVal)
        {
            Console.WriteLine("guessed value is smaller than target value");
            lowestVal = guessVal;
        }
        else if (guessVal > targetVal)
        {
            Console.WriteLine("guessed value is bigger than target value");
            highestVal = guessVal;
        }
        else
        {
            Console.WriteLine("you guessed right, target value is {0}", targetVal);
            break;
        }
    }
}

```

(Figure 14)

Simulation

We can take a simulation to create a human by code.

First of all, we need to know that when creating a specific person, we should not use the specific person as a coding template, but abstract the specific person into a concept and use it as a template. Because, if we create a code template for a specific person, then the code will have low reusability, but if we create a template for a concept, then the reusability of the code will be very high. Why emphasize reusability? Because computer resources are precious and should be cherished.

As we all know, humans have height, weight, eyes and hair, etc. so we can create a class as shown in Figure 15, add a constructor, and when we create a specific person, we can provide the necessary fields for this person.

```

1 reference
public class Human{
    1 reference
    private string _Name;
    1 reference
    private double _Height;
    1 reference
    private double _Weight;

    1 reference
    private List<Eye> _Eyes;

    1 reference
    private Hair _Hair;

    0 references
    public Human(string name,double height,double weight,List<Eye> eyes, Hair hair){
        _Name = name;
        _Height = height;
        _Weight = weight;
        _Eyes = eyes;
        _Hair = hair;
    }
}

2 references
public class Eye{
    0 references
    private Color _Color;
}

2 references
public class Hair{
    0 references
    private Color _Color;
}

```

(Figure 15)

Human can walk, dance, and speak, etc.

```

0 references
public Human(string name,double height,double weight,List<Eye> eyes, Hair hair){
    _Name = name;
    _Height = height;
    _Weight = weight;
    _Eyes = eyes;
    _Hair = hair;
}

0 references
private void Walk(){
    Console.WriteLine($"{_Name} is walking");
}

0 references
private void Dance(){
    Console.WriteLine($"{_Name} is dancing");
}

0 references
private void Speak(){
    Console.WriteLine($"{_Name} is Speaking");
}
}

```

(Figure 16)

Now, we can build a human to do somethings.

```
0 references
public void Main(string[] args){
    string name = "John";
    double height = 170.00;
    double weight = 140.00;
    List<Eye> eyes = new List<Eye>(2);
    eyes.Add(new Eye(Color.Brown));
    eyes.Add(new Eye(Color.Brown));
    Hair hair = new Hair(Color.Black);
    Human John = new Human(name,height,weight,eyes,hair);
    John.Walk();
    John.Dance();
    John.Speak();
}
```

(Figure 17)

Console will output this.

```
75448@FlyBird MINGW64 /d/code_2024/past-future/deakin/sit771/4.3C/RobotDodge
$ skm dotnet run Human.cs
John is walking
John is dancing
John is Speaking
```