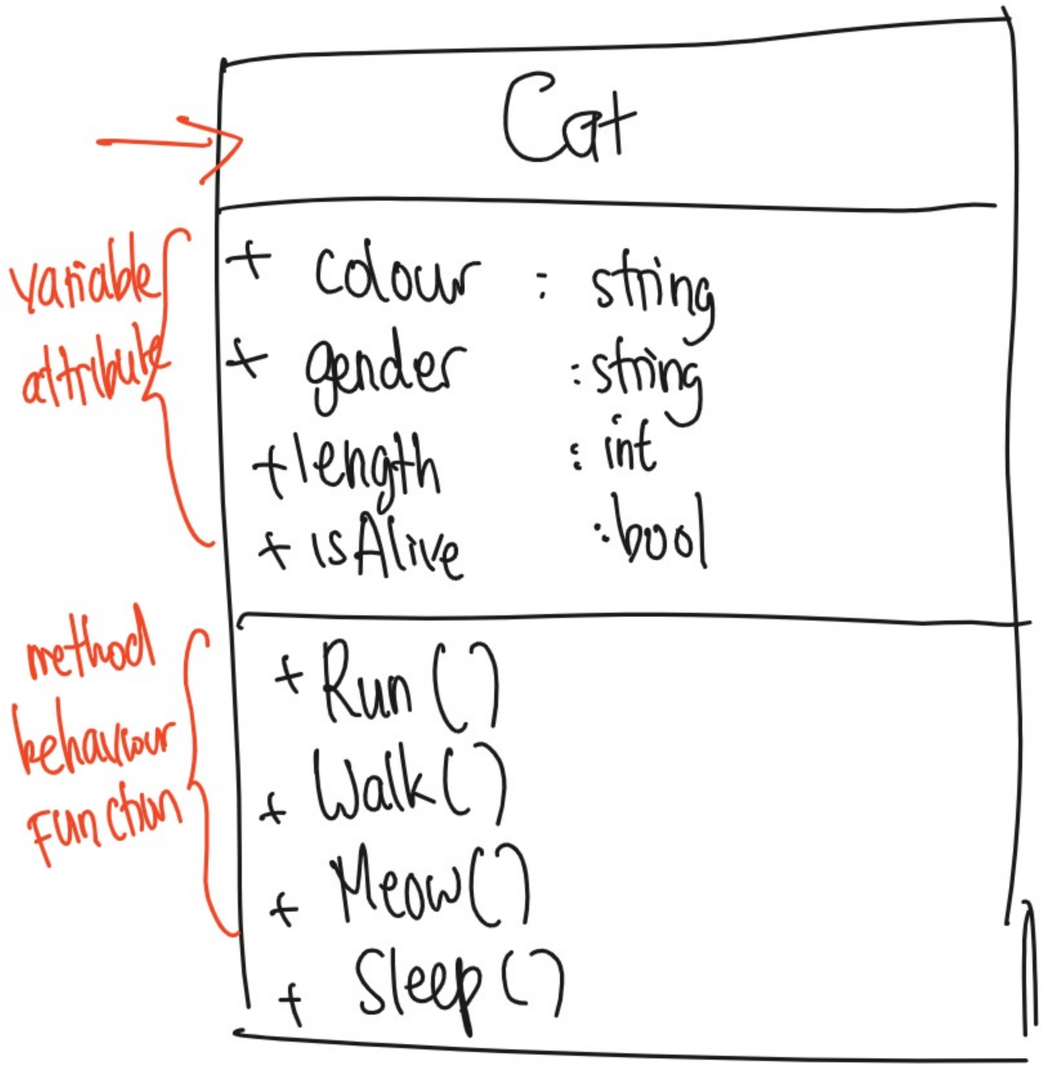
Environment Setup

1. IDE : Integrated Development Environment
    a. Visual Studio Code (Free)
        https://code.visualstudio.com/Download
        Extensions :
            i. C#
            ii. .NET Test Explorer
            iii. vscode solution explorer
            iv. Live Share
    b. Visual Studio (Paid)
    c. Jetbrains Rider (Paid)
    d. LINQPad (Free)
        https://www.linqpad.net/Download.aspx
2. Software Development Kit (check dotnet --version in Terminal)
    Microsoft .NET 8 (Long Term Support)
3. Version Control (Git / GitHub Desktop / SourceTree)


Cat
+ colour : string
+ gender : string
+ length : int
+ isAlive : bool
+ Run ()
+ Walk ()
+ Meow ()
+ Sleep ()
variable attribute
method behaviour function

```
class AnggoraCat                    → PascalCase
{
    public string name;             → camelCase
    public bool isAlive;
    public void Jump()              → PascalCase
    {

    }
}
```

```
static void Main()
{
    Cat cinamon = new Cat();
}
```

cinamon → camelCase

Employee

name : string
id : int
email : string

Work() : void
rest() : void

```
class Employee
{   public string name;
    public int id;
    public string email;

    public void Work()
    {
    }
    public void Rest()
    {
    }
}
```

```
class Program
{
    static void Main()
    {
        Employee emp
            = new Employee();
        emp.name = "Dion";
        emp.id = 69;
        emp.email = "dion@fmk";

    }
}
```

```
class Program
{
    static void Main()
    {
        Employee emp = new Employee();
    }
}
```

constructor

```
class Employee
{
    public string name;
    public int id;
    public string email;

    public Employee()
    {
    }

    public void Work()
    {
        ....
    }
    public void Rest()
    {
        ....
    }
}
```

default constructor

constructor = create object
create instance

```
class Employee
{   public string name;   <-- wajib
    public int id;        <-- wajib
    public string email;  <-- optional

        public Employee( string name1 , int id )
        {this.name = name;
         this.id = id;
        }

    // method ------

}
```

name = name1

Constructor

```
class Program {
    static void Main() {
        Employee emp = new Employee("Dion",69);
        Console.WriteLine(emp.name);
        Console.WriteLine(emp.id);

        emp.email = "dion@fmlx.com";
        Console.WriteLine(emp.email);
    }
}

class Employee {
    public string name;
    public string email;
    public int id;

    //Constructor
    public Employee(string name, int id) {
        this.name = name;
        this.id = id;
    }
}
```

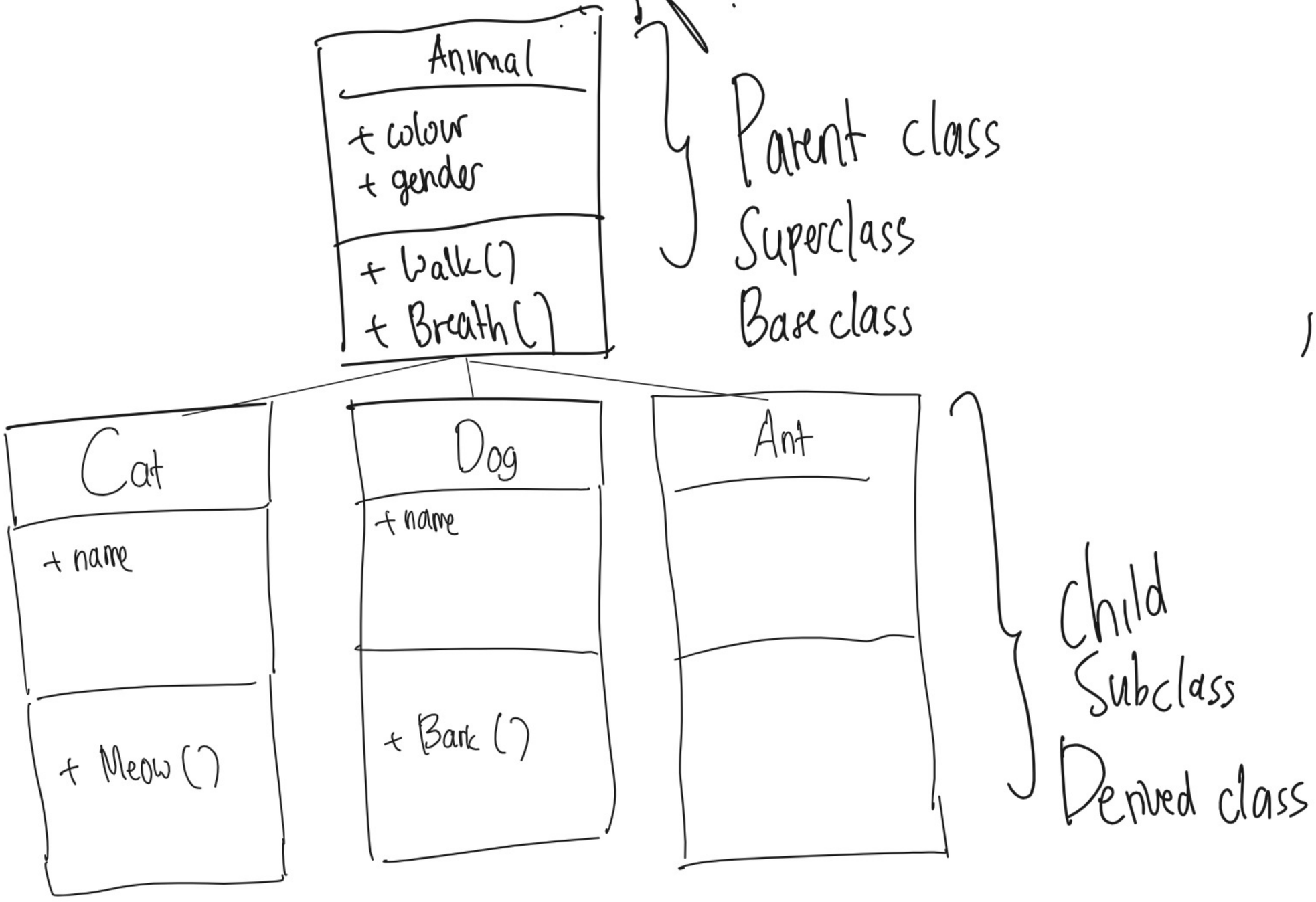
Code block

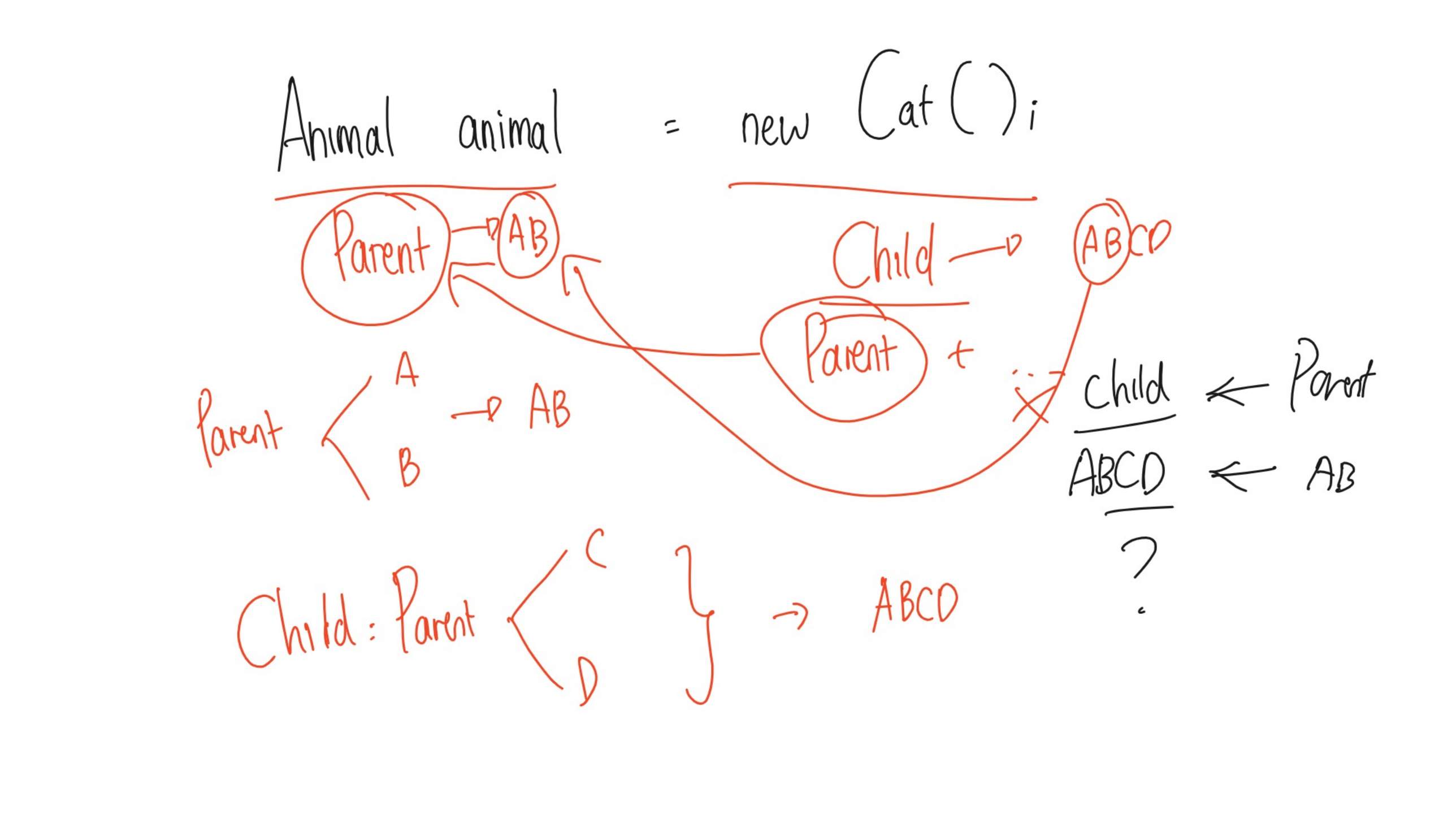
```
//Overloading
//Method/Constructor with same name, but different parameter
class Program {
    static void Main() {
        Employee emp = new Employee("Dion",69);
        Console.WriteLine(emp.name);
        Console.WriteLine(emp.id);
        Employee emp2 = new Employee("Dion",69,"dion@fmlx.com");
        Console.WriteLine(emp.email);
    }
}

class Employee {
    public string name;
    public string email;
    public int id;

    //Constructor (name , id)
    public Employee(string name, int id) {
        this.name = name;
        this.id = id;
    }
    //Constructor (name, id, email)
    public Employee(string name, int id, string email)
        {
        this.name = name;
        this.id = id;
        this.email = email;
    }

    //Add additional constructor
    //Parameterless Constructor
    public Employee()
    {
        id = 0;
        name = "default";
        email = "default@fmlx.com";
    }
}
```

OOP → Object Oriented Programming

1. Inheritance
2. Encapsulation
3. Polymorphism
4. Abstraction

① Inheritance




Animal animal = new Cat();
Parent
AB
Child
ABCD
Parent
t
Parent
A
B
AB
Child : Parent
C
D
ABCD
Child ← Parent
ABCD ← AB
?

```
int a = 2;
int b = a;
b += 3;
```

→ (b = b + 3)

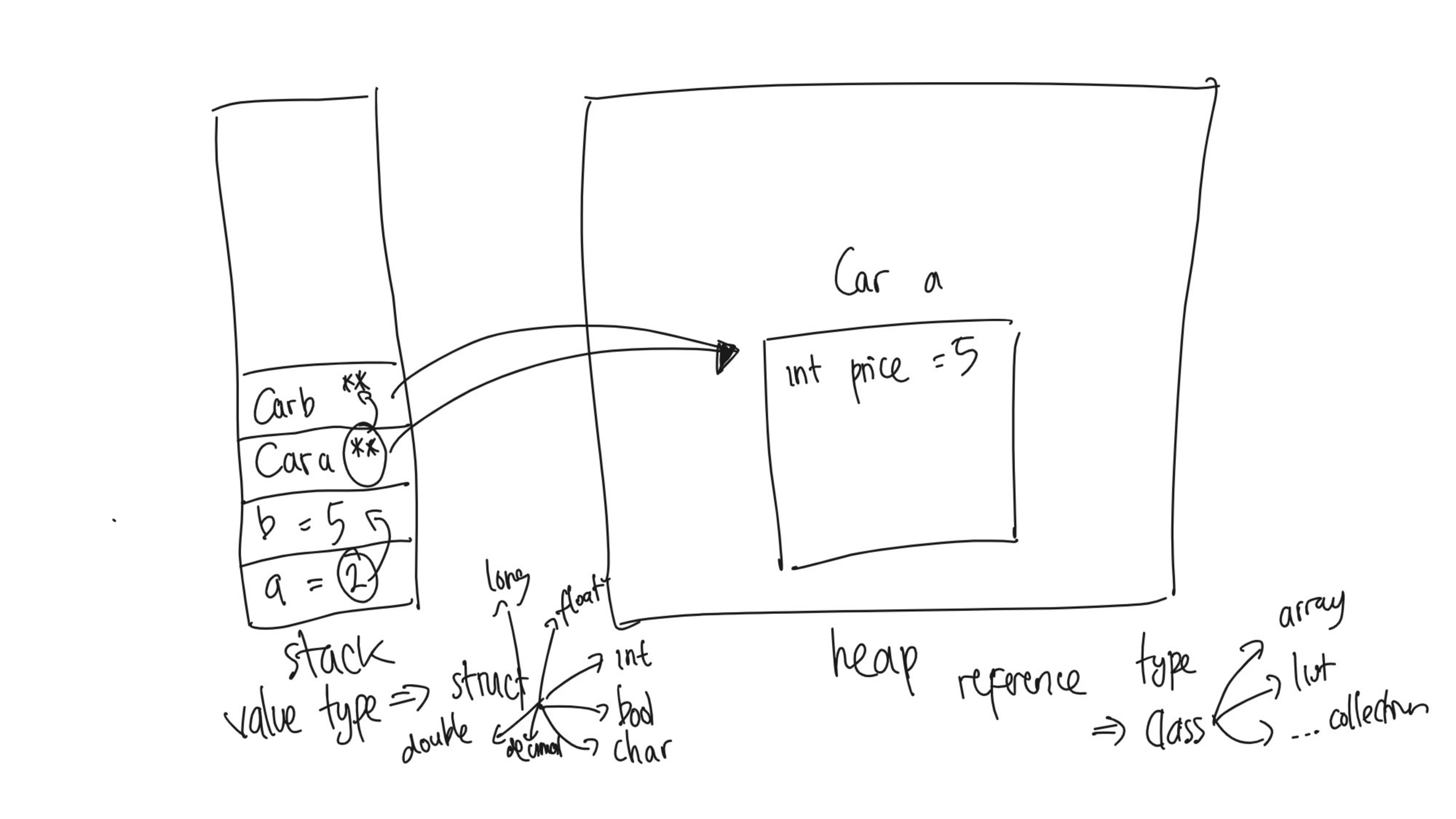
a	?	2	5
b	?	5	5
		5	1

```
class Car
{
    public int price;
    public Car(int x)
    {
        price = x;
    }
}
```

```
Car a = new Car(2);
Car b = a;
b.price += 3;
```

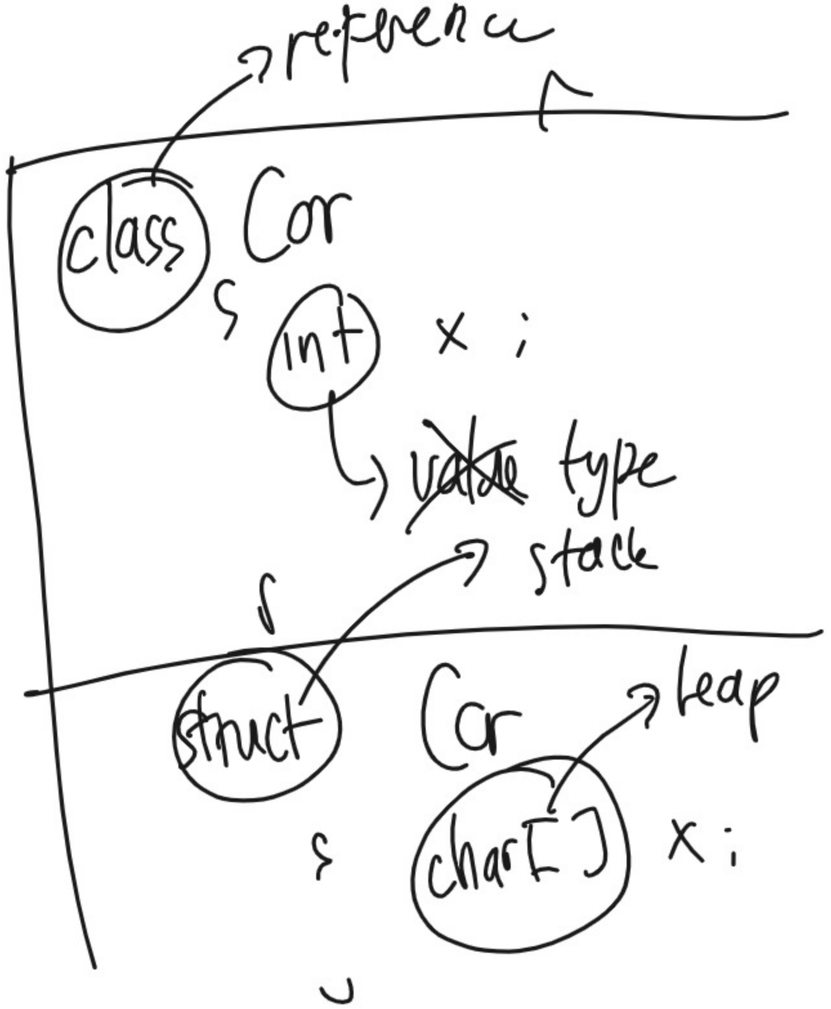
a.price ? 2 → 5

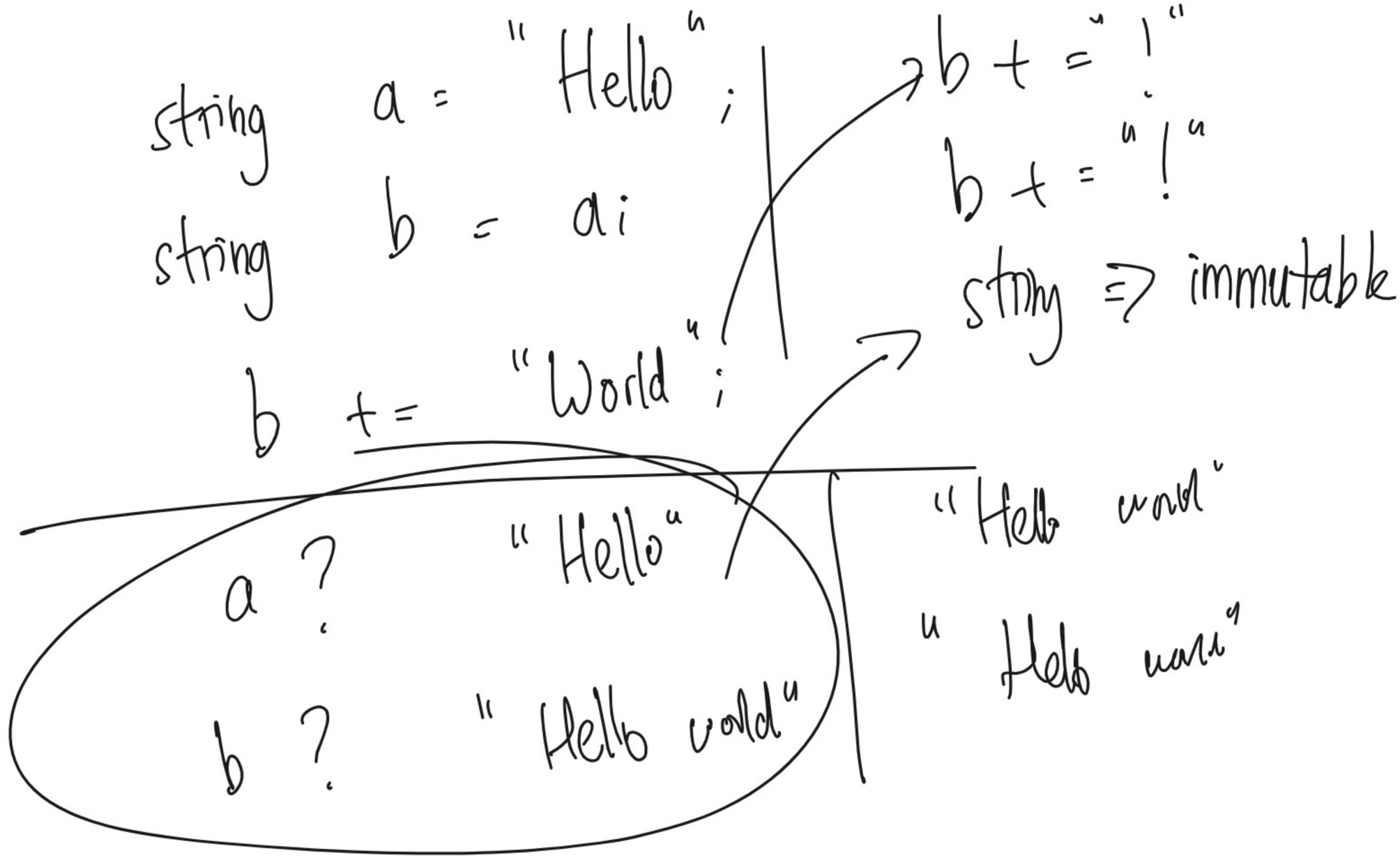
b.price ? 5 → ~~7~~ → 5 → 2

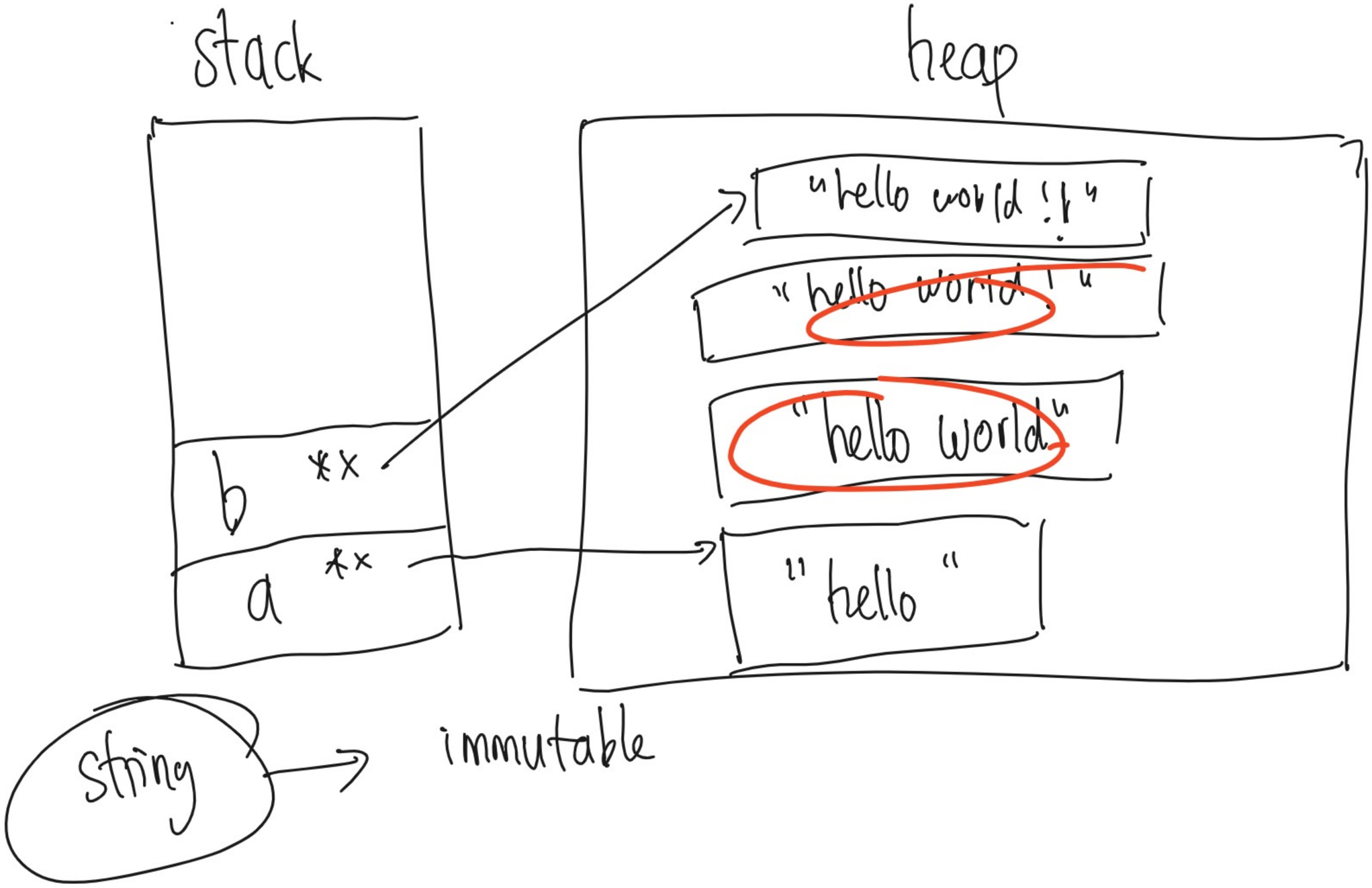

Carb **
Cara **
b = 5
a = 2
stack
Car a
int price = 5
heap
value type => struct
long
float
int
bool
char
decimal
double
reference type
=> class
array
list
... collection

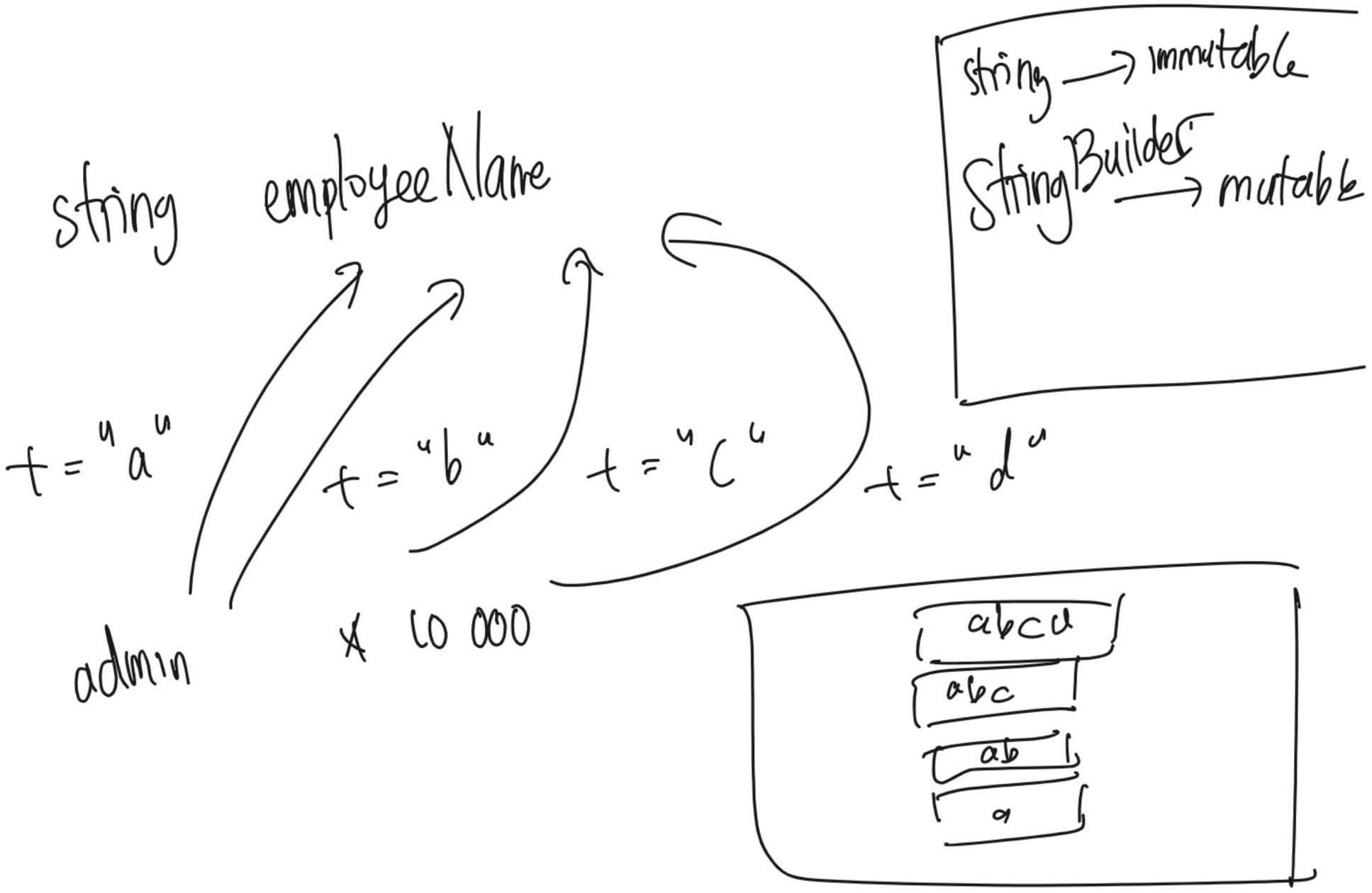
struct → - property / variable must be assigned
- x
- stack
- value type

class → - x
- inheritance
- heap
- reference type




string a = "Hello";
string b = a;
b += "World";
b += "!"
b += "!"
string => immutable
a ? "Hello"
b ? "Hello world"
"Hello world"
"Hello world"


stack
heap
b
**x
a
**x
"hello world!"
"hello world"
"hello"
String
immutable


string employeeName
t = "a"
t = "b"
t = "c"
t = "d"
admin
X 10 000
string → immutable
StringBuilder → mutable
abcd
abc
ab
a

sdution = dotnet new sln

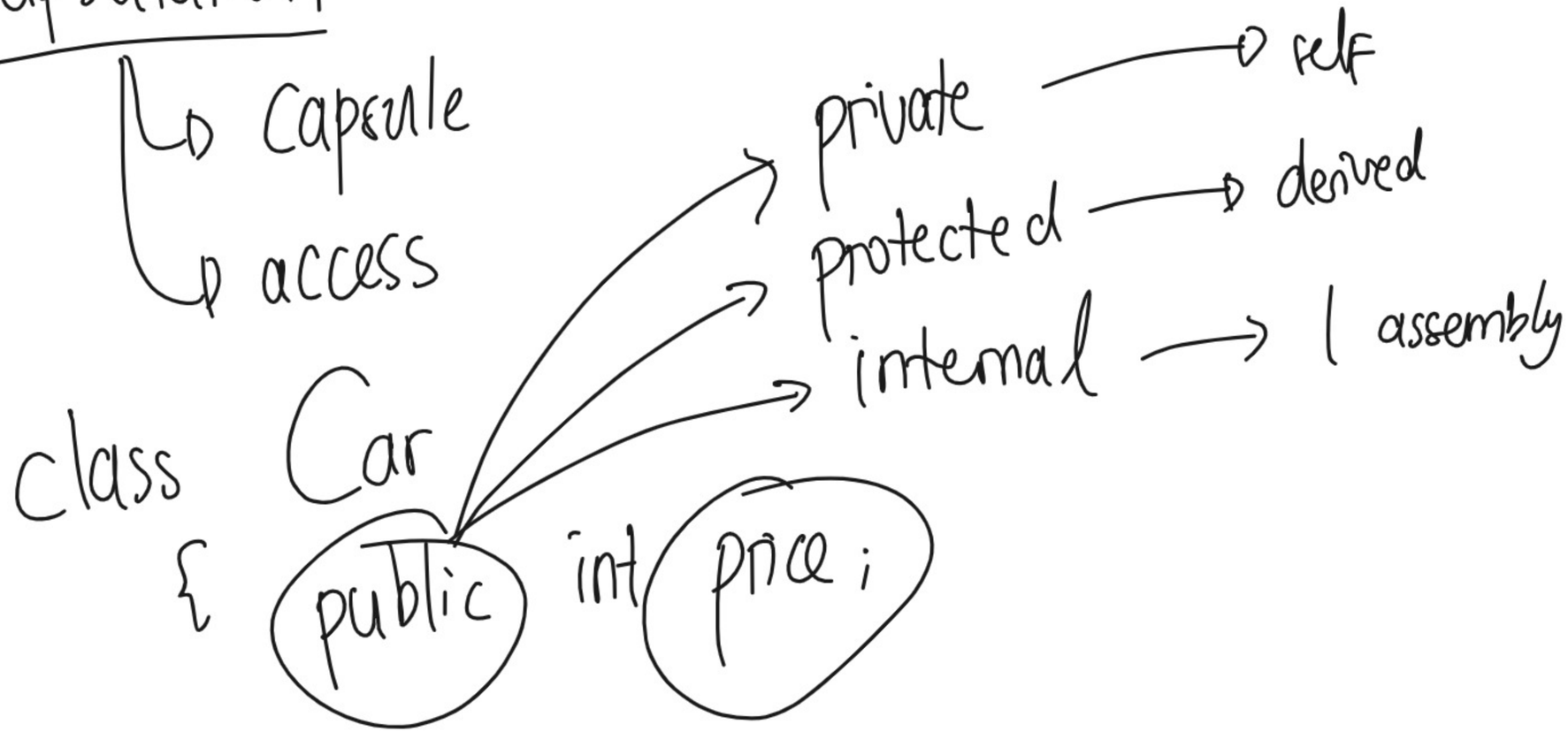
Console = dotnet new console

class library = dotnet new classlib

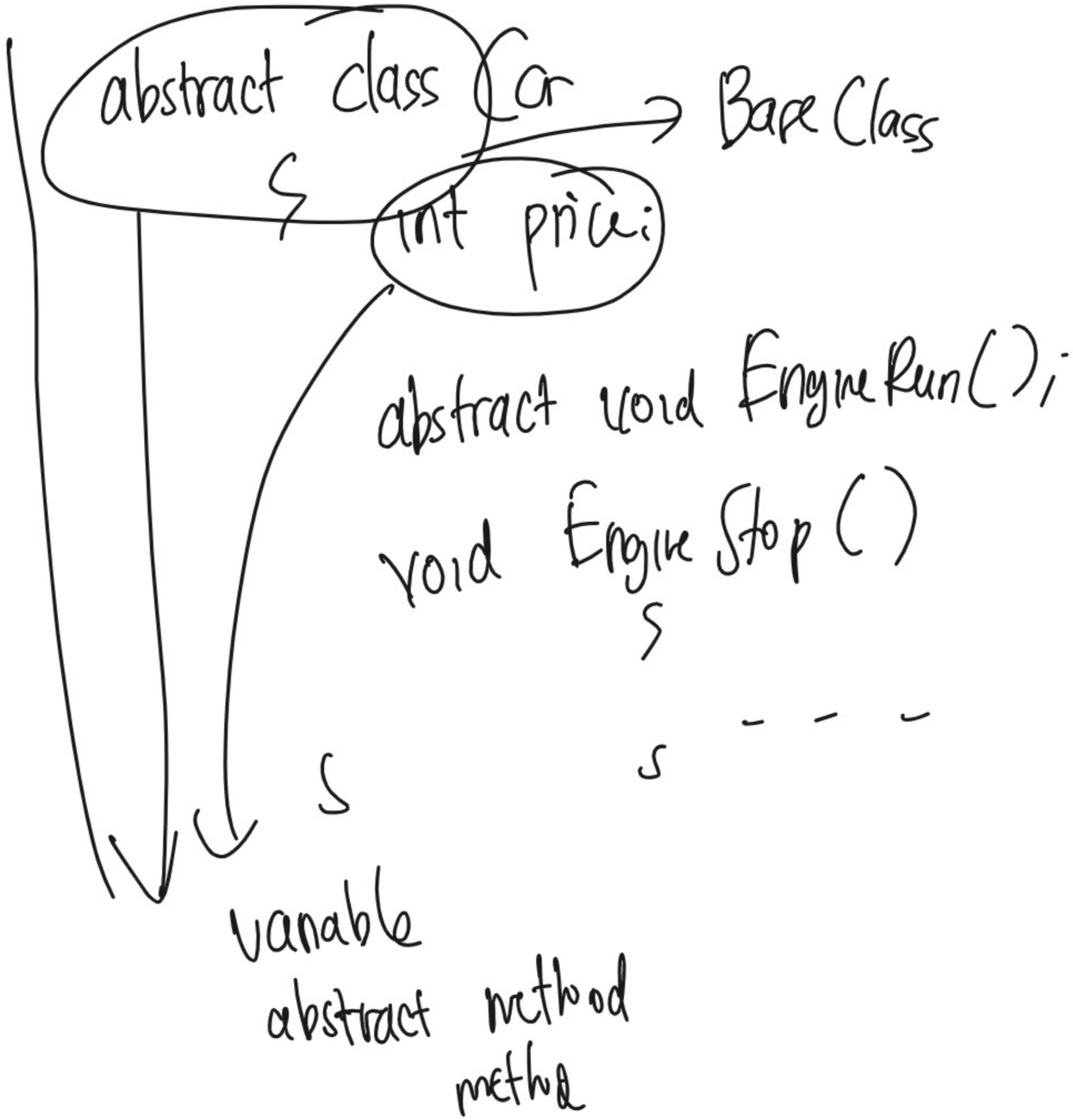
.gitignore = dotnet new gitignore

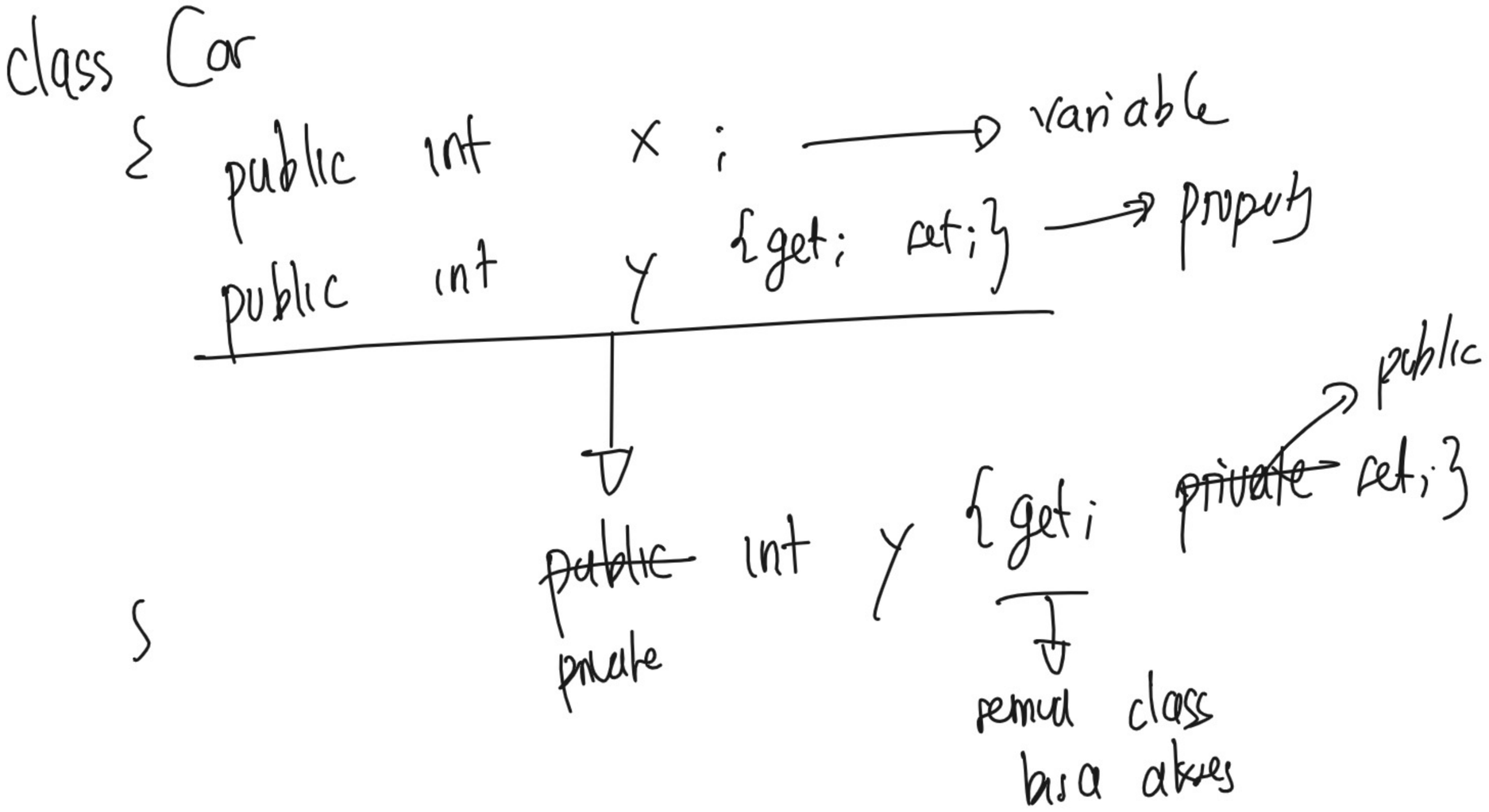
(2) Encapsulation

- capsule
- access



```
class Car
{ int price;

  void EngineRun()
  {
    ----
  }
  void EngineStop()
  {
    ----
  }
}
```




class Car
{
public int x ;
variable
public int y {get; set;}
property
public
~~public~~ int y {get; ~~private~~ set;}
private
}
sama class
bisa akses

abstract class Car

{ public int x;

public int y {get; set;}

public abstract Engine Run();

public void Engine Stop() { ... }

}

→ single inheritance

class Truck : Car

{

override

}

interface ICar

{ int y {get; set;}

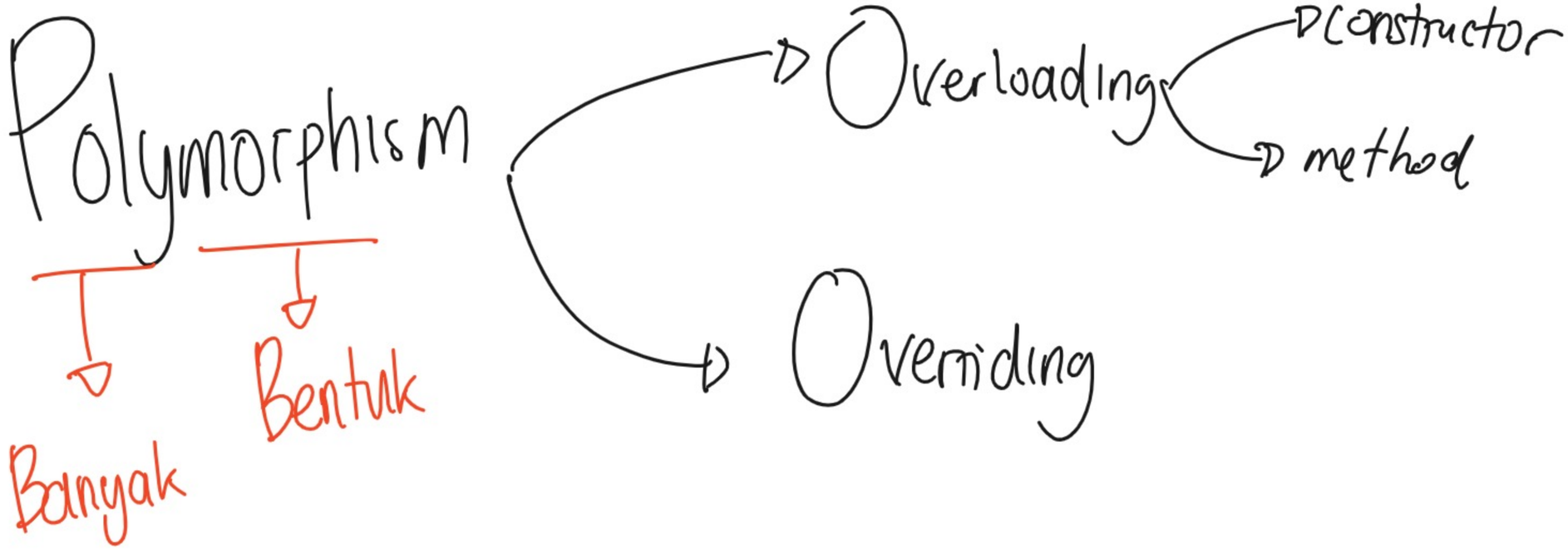
Engine Run();

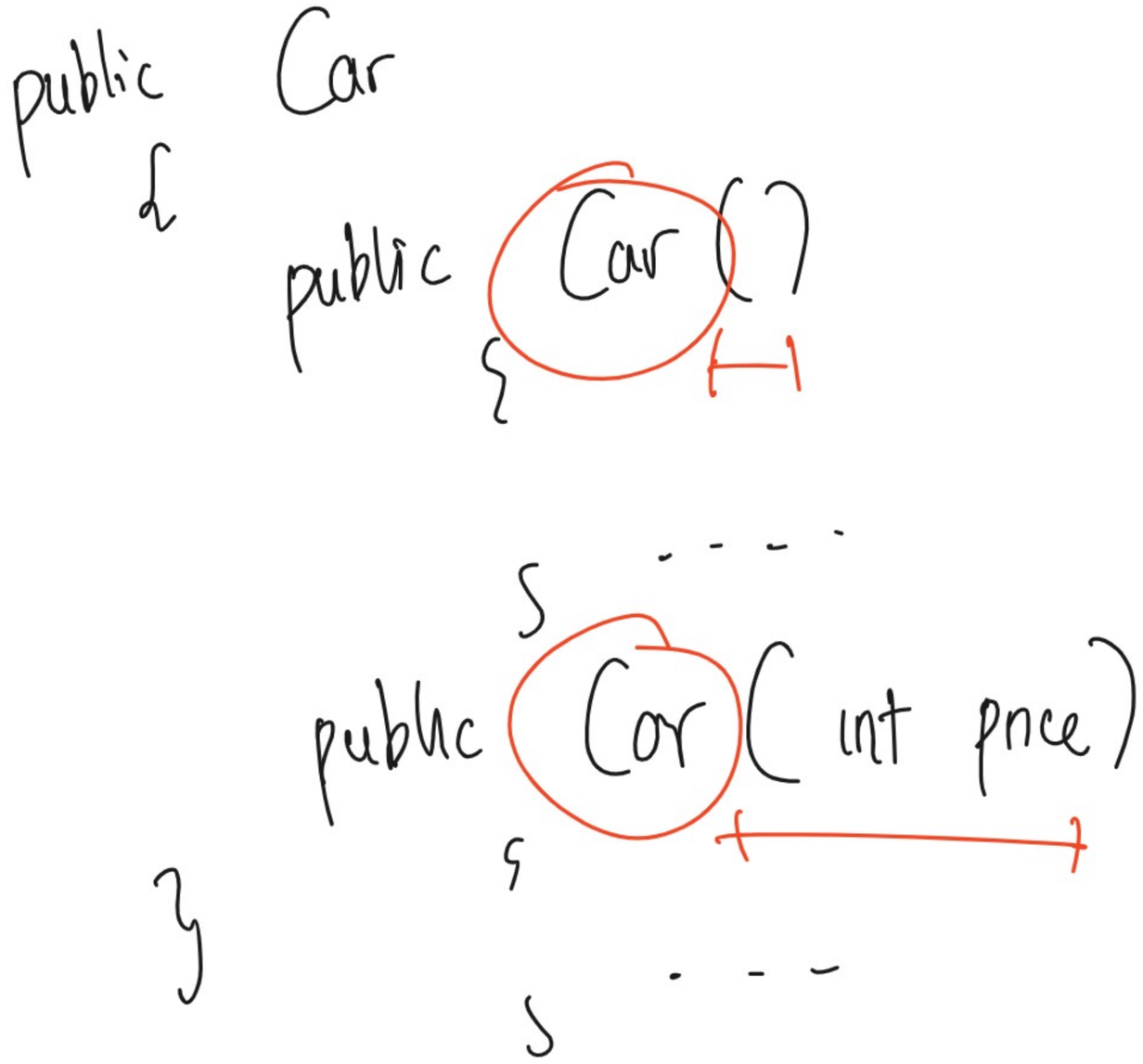
Engine Stop();

}

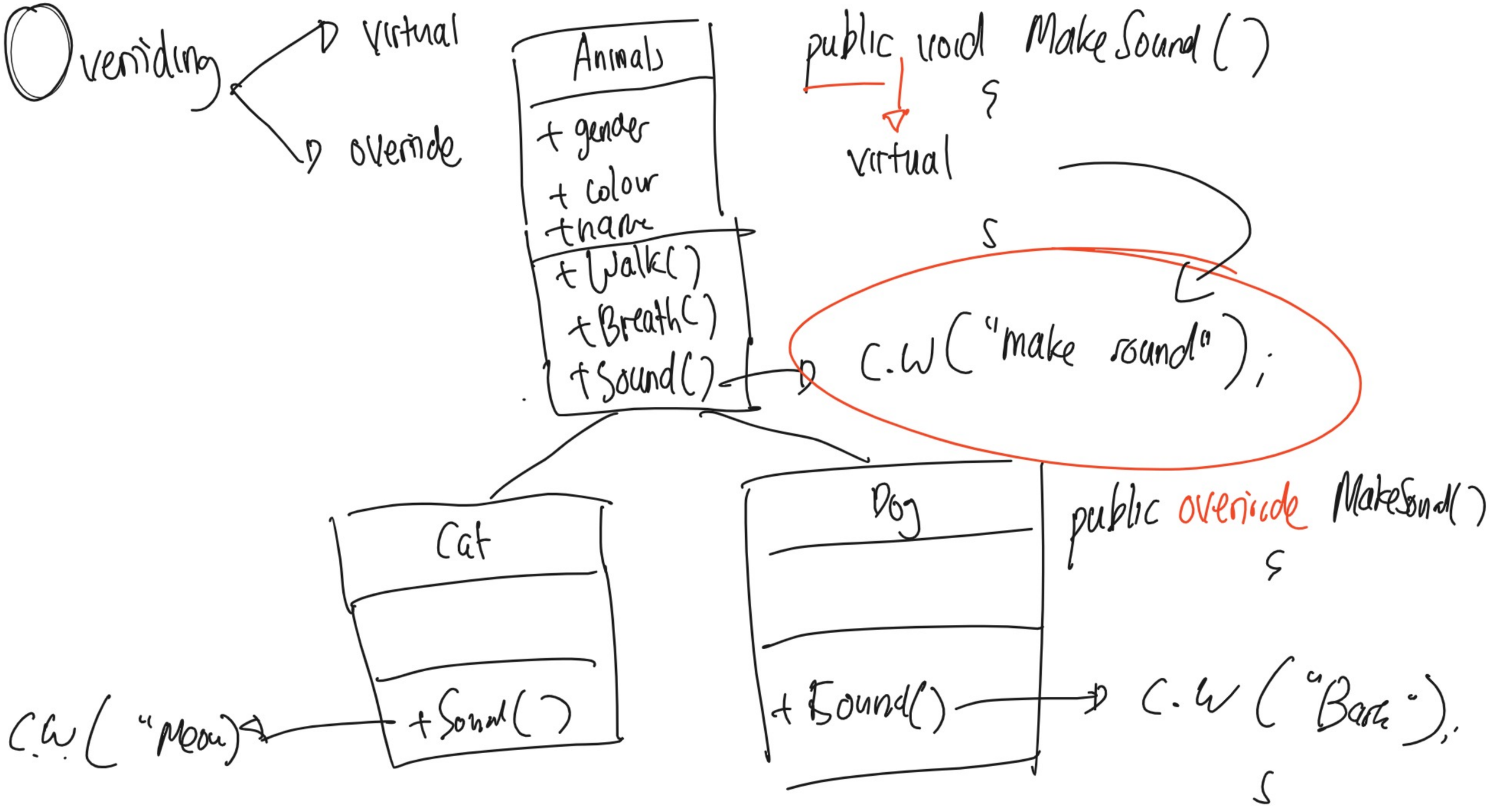
→ multiple inheritance

class Truck : ICar, I...


Polymorphism
Banyak
Bentuk
Overloading
Constructor
method
Overriding

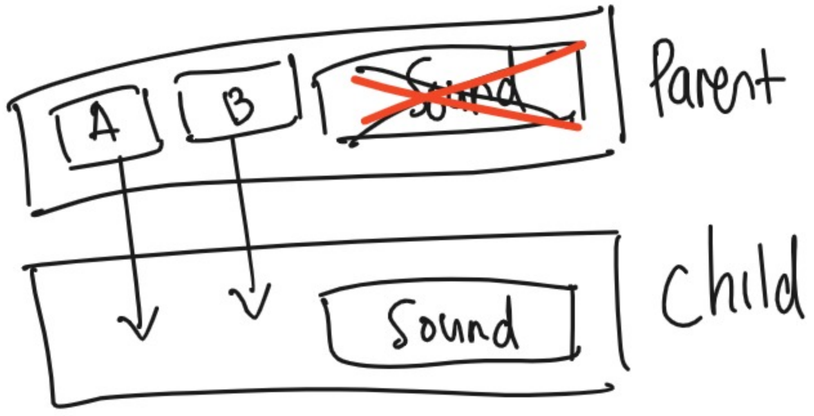

public Car
{
public Car()
{
}
.....
public Car(int price)
{
}
.....
}

```
public Cat
{
    public void Eat() { .... }
    public void Eat(string x) { .... }
    public int Eat() { .... }
    public int Eat(int ...)
}
```


Overriding
virtual
override
Animals
+ gender
+ colour
+ name
+ Walk()
+ Breath()
+ Sound()
public void MakeSound()
{
virtual
}
C.W("make sound");
Cat
+ Sound()
C.W("Meow");
Dog
+ Sound()
public override MakeSound()
{
C.W("Bark");
}

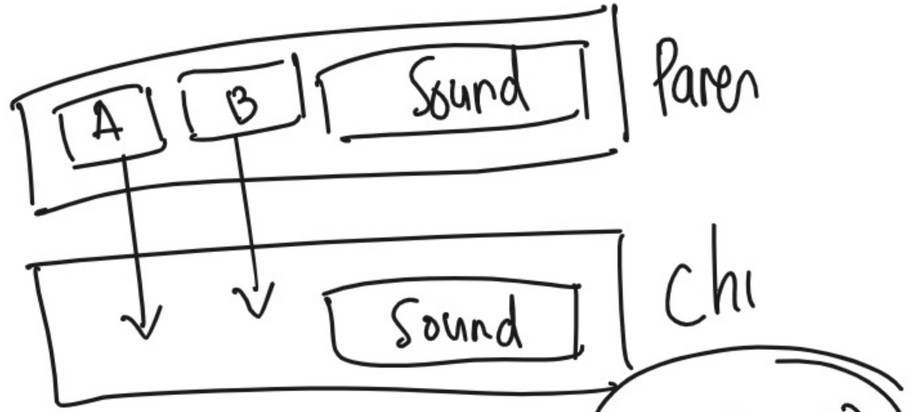
Overriding

```
class Animal
{  virtual Sound()
   {  C.W("animal");
   }
}
class Cat : Animal
{  override Sound()
   {  C.W("Meow");
   }
}
```

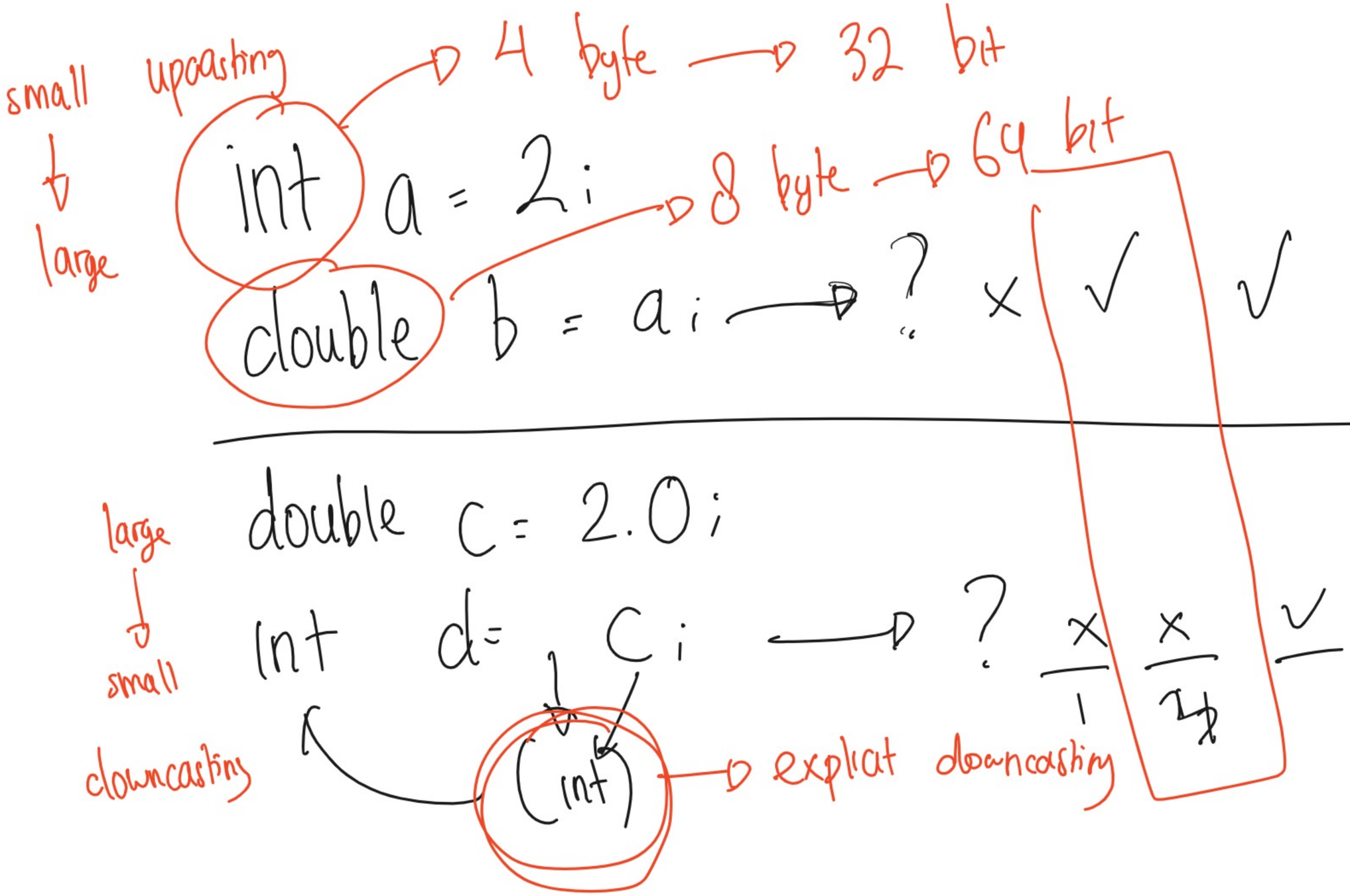


Method Hiding

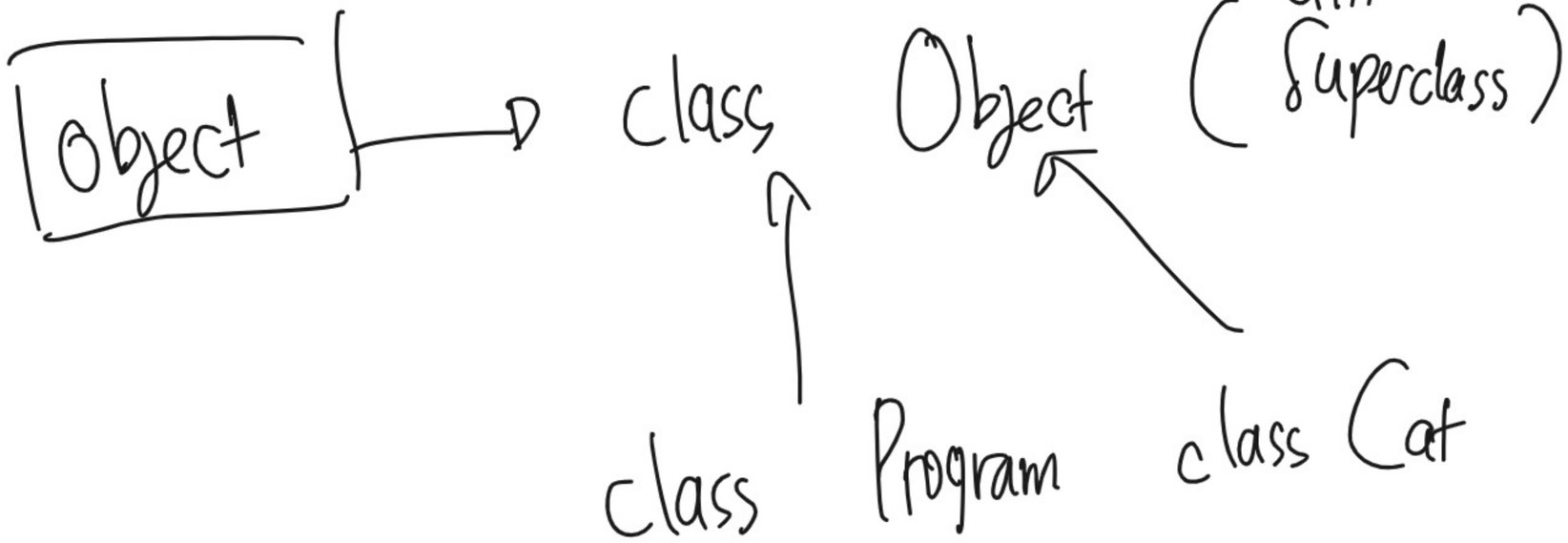
```
class Anima
{  virtual Sound()
   {  C.W("animal");
   }
}
class Cat : Animal
{  Sound()
   {  C.W("Meow");
   }
}
```



Animal an = new Cat();


small
large
upcasting
int a = 2;
4 byte → 32 bit
double b = a;
8 byte → 64 bit
? x ✓ ✓
large
small
double c = 2.0;
int d = c;
? x x ✓
1 4
downcasting
(int)
explicit downcasting

Boxing & unboxing




```
object obj = new Object()
    object obj = 3;
        object obj2 = true;
```

```
class Human
{
    private static int counti;
    private int _balance;

public int GetCount()
{
    return count;
}

public static int GetCount()
{
    return count;
}

}
```

✓ ✓

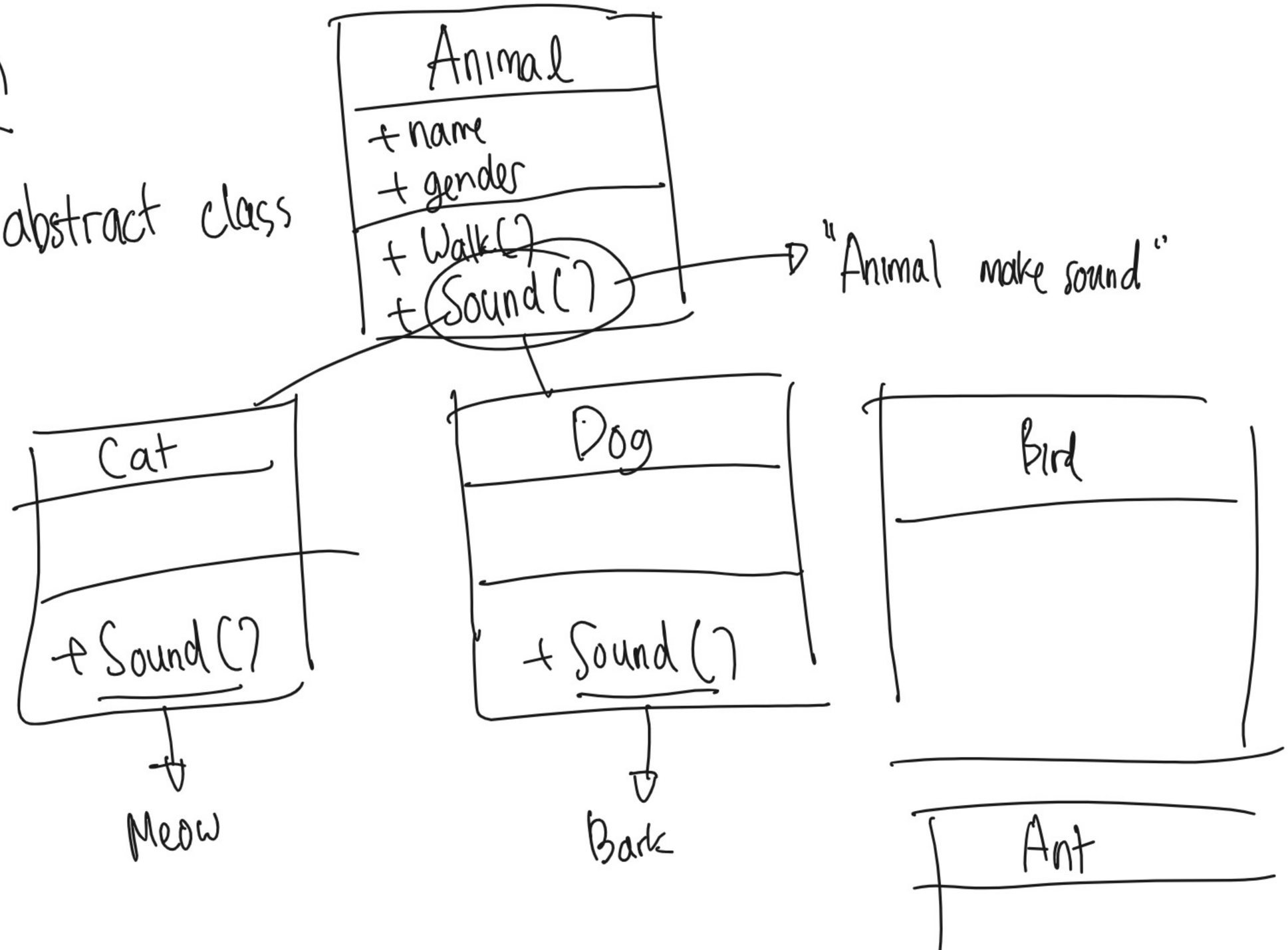
```
public int GetBalance()
{
    return _balance;
}
```

```
public static int GetBalance()
{
    return _balance;
}
```

✗

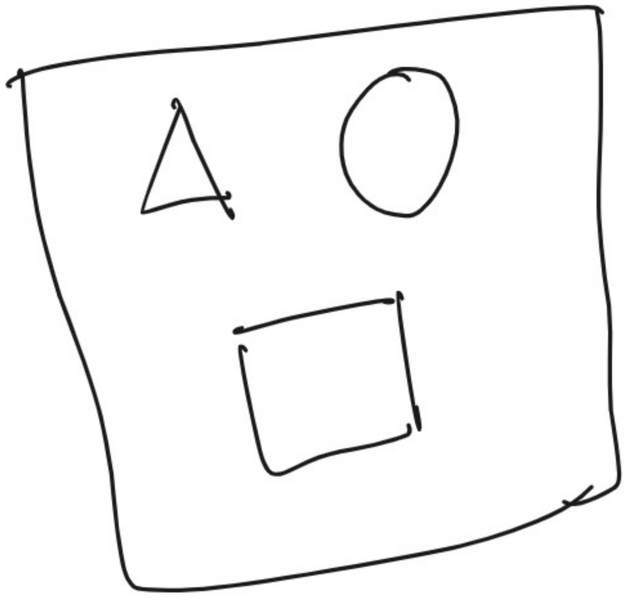
④ Abstraction

↳ abstract class



Base class

class



- have all implementation
- Single inheritance

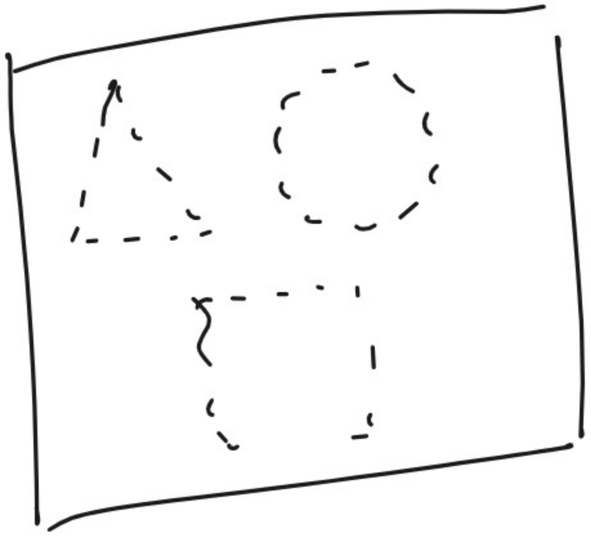
abstract class

- can have abstract method
- as base class

parent class

- single inheritance

interface



- contract

class Parent / abstract class Parent

{

(A) public void Kuliah()
{ "UGM".Dump(); } || child → method hiding

(B) public virtual void Kuliah()
{ "UGM".Dump(); } || child → override

(C) public abstract void Kuliah(); || child → override

}

class Child : Parent
{ public override void Kuliah()
{ base.Kuliah();
"BINUS".Dump(); }

- Properties
- Interfaces (ICard, ICar, dsb) / Contract
- Enum
- Generic
- Generic Constraint
- Ref In Out

Property / Variable Method

```
public int x;
```
← variable

```
public int x {get; set;}
```
← property

Interface

```
public interface IAnimal
{
    void Walk();
    void Sound();
    void Breath();
}
```

- methode
- property

public

do not have implementation

```
interface IJump
{
    void Jump();
}
```

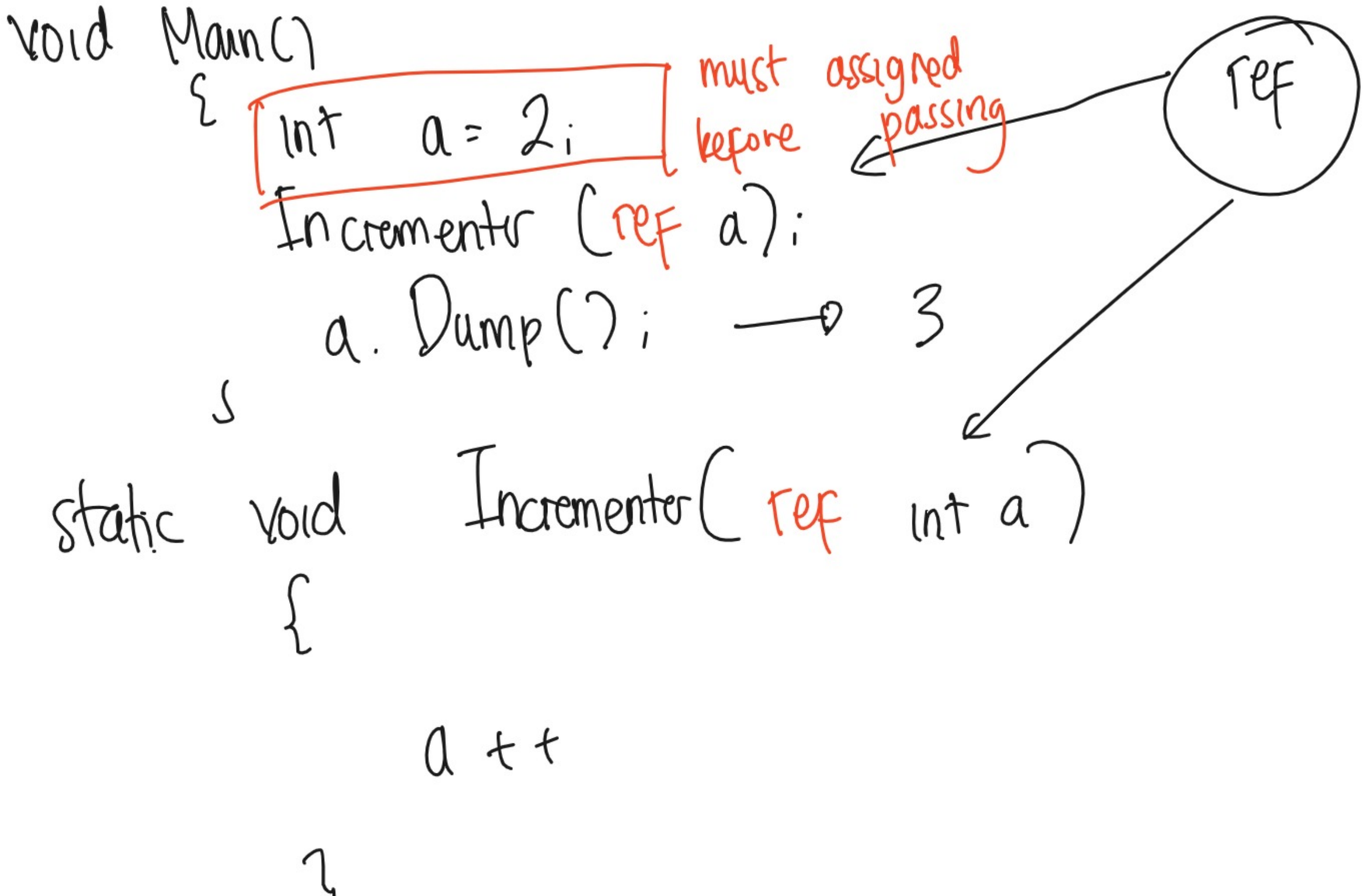
```
class Cat : IAnimal, IJump
{
    public void Walk()
    {
        ....
    }
    public void Sound()
    {
        ....
    }
    public void Breath()
    {
        ....
    }
    public void Jump()
    {
        ...
    }
}
```

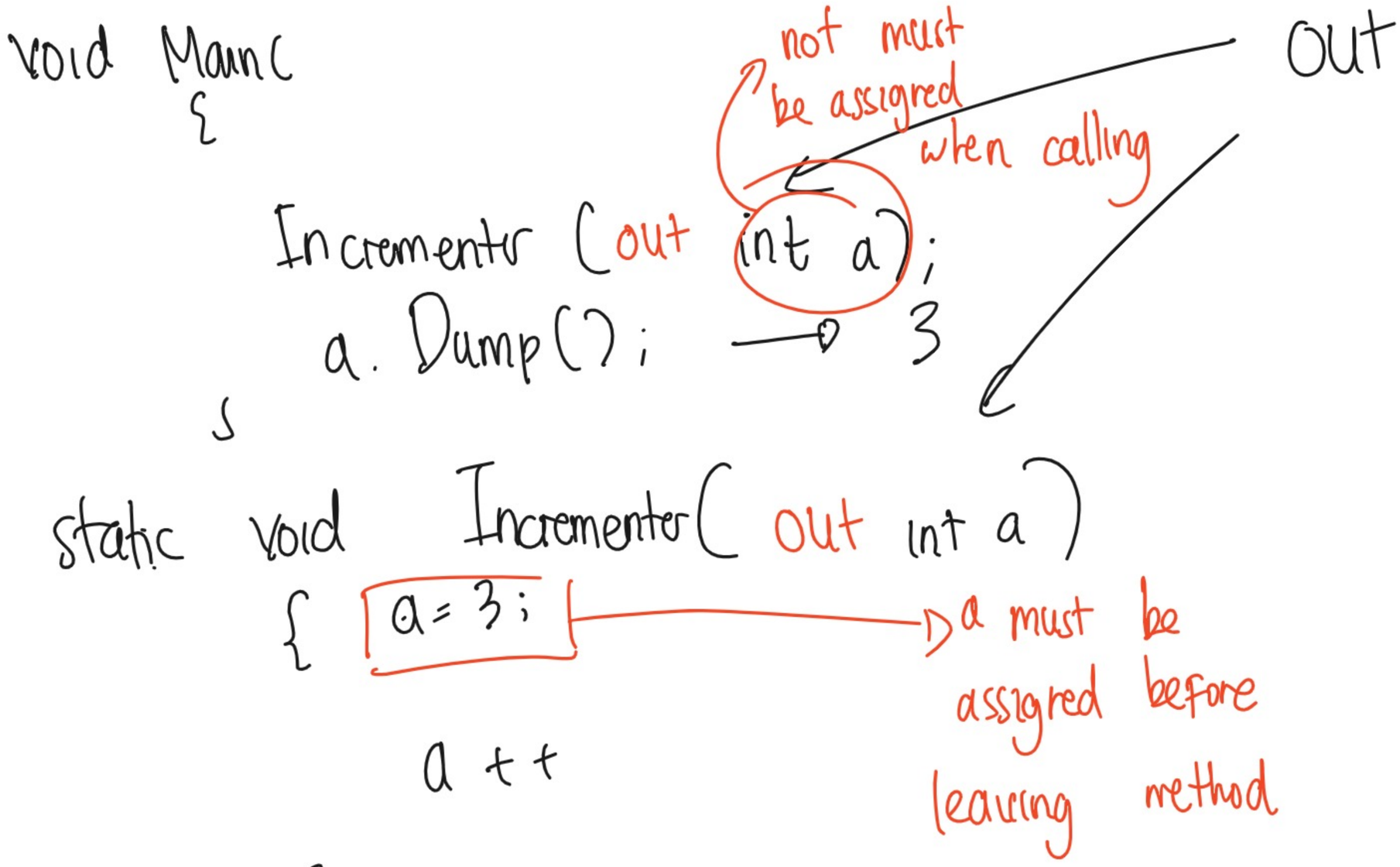
no need for overide

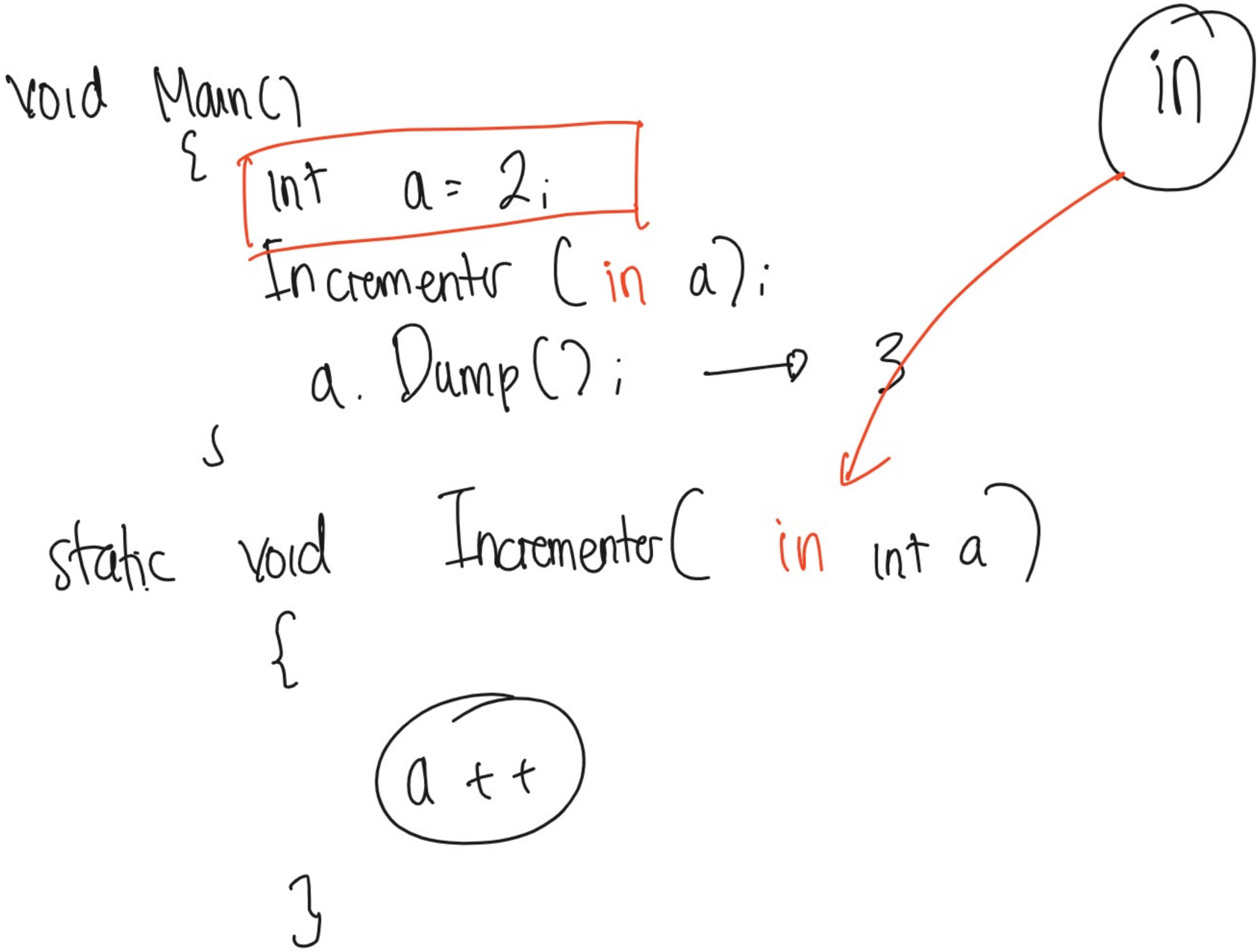
```
{ int a = 2;                                   ①
  Incrementer(a);
  a.Dump(); ?
}

static void Incrementer(int a)        2
{
  a++; → 3
}
```

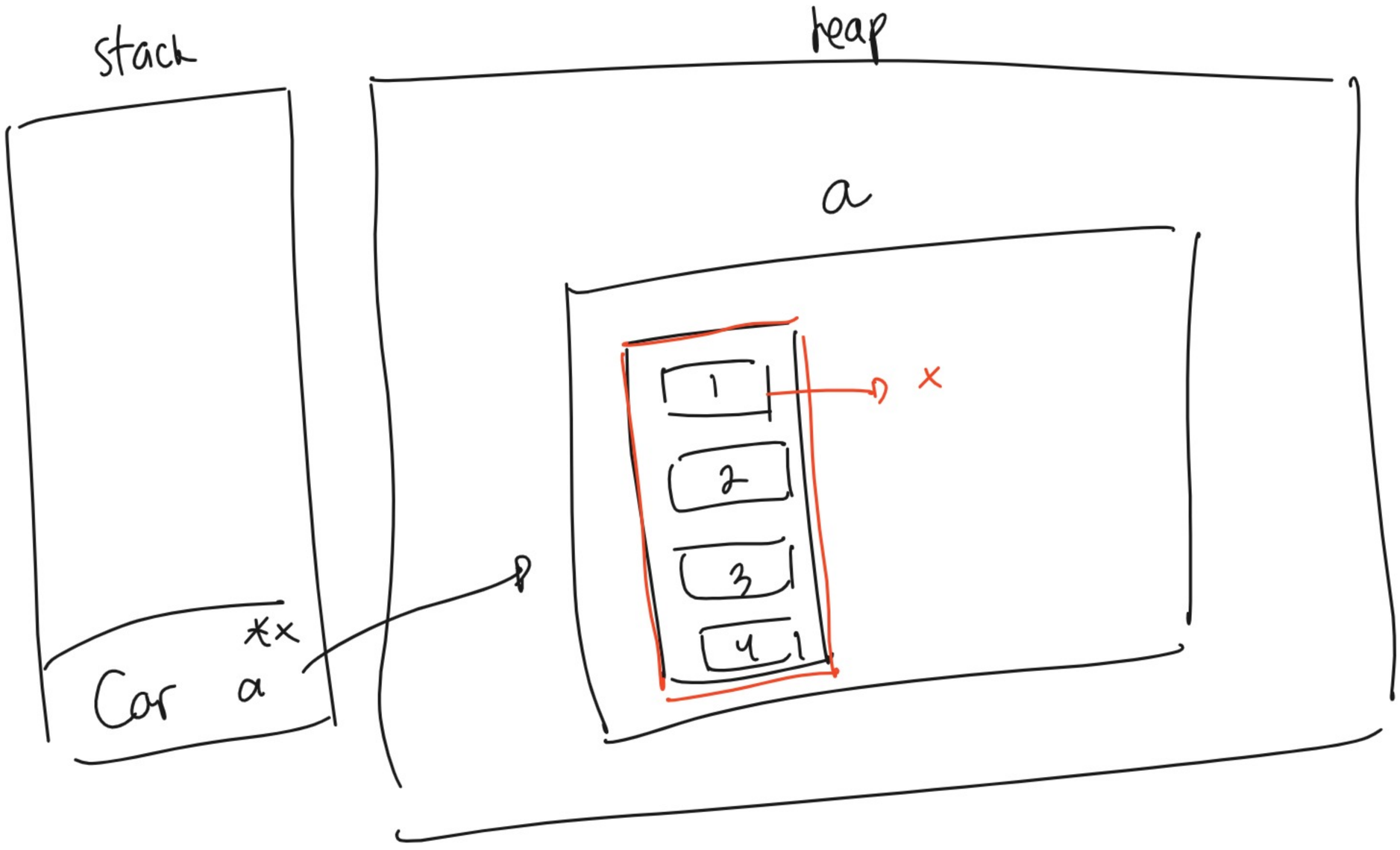
2 | 3
3 | 3

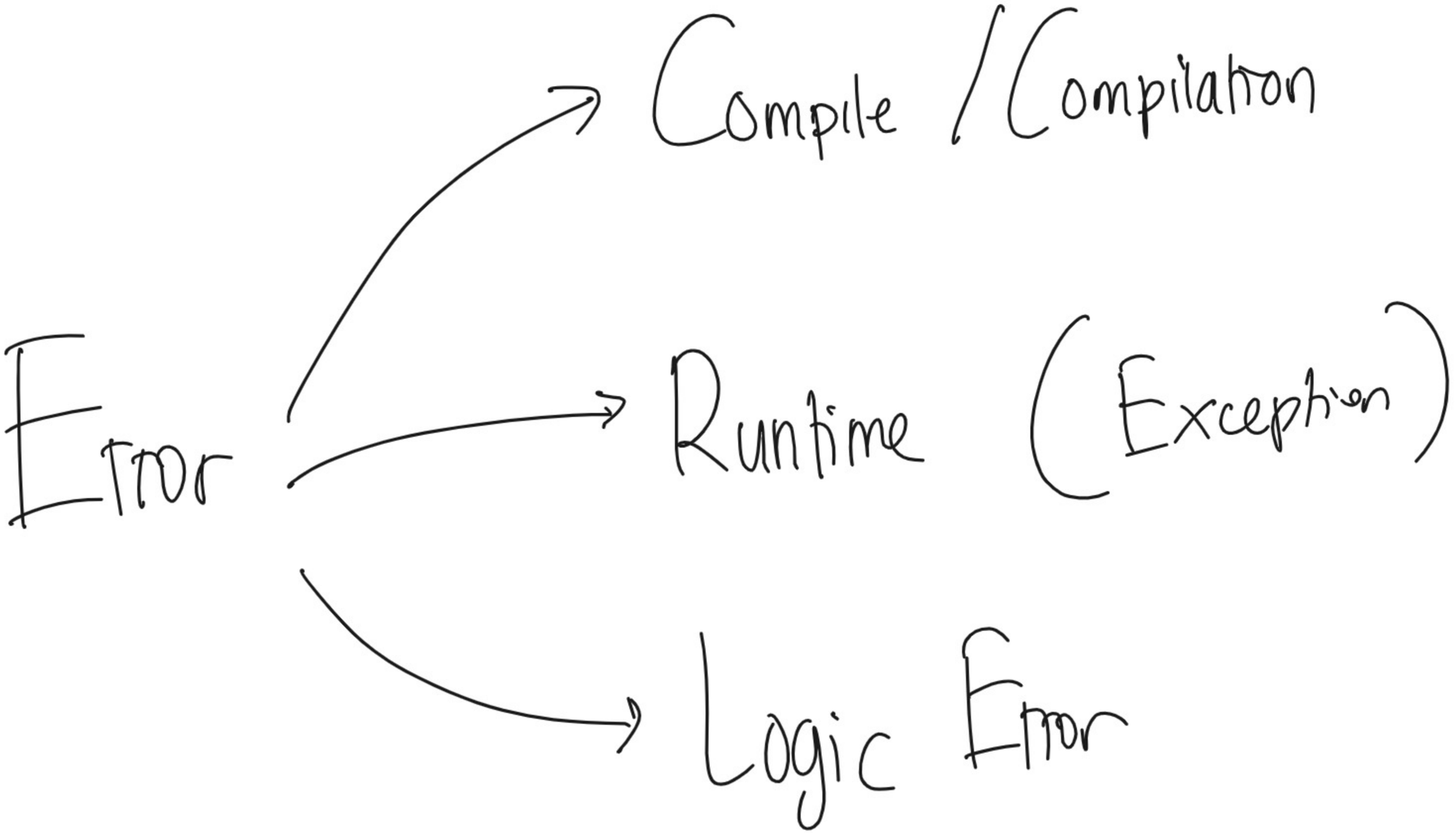
```
{ Car a = new(2);                              ②
  Incrementor(a);
  a.price.Dump(); ?
}

static void Incrementer(Car a)
{
  a.price++; 3 | 3
             3 | 3
}
```


void Main()
{
int a = 2;
must assigned before passing
ref
Incrementer (ref a);
a.Dump(); → 3
}
static void Incrementer(ref int a)
{
a++
}


void Main()
{
Incrementer (out int a);
a.Dump();
}
not must be assigned when calling
out
3
static void Incrementer(out int a)
{
a = 3;
a must be assigned before leaving method
a++
}


void Main()
{
int a = 2;
Incrementer (in a);
a.Dump();
}
static void Incrementer(in int a)
{
a++
}
in
3


stack
heap
a
1
2
3
4
x
*x
Car
a


Error
Compile / Compilation
Runtime (Exception)
Logic Error

Delegate

- must same signature
- public delegate void MyDelegate (int x)

void: return
MyDelegate: delegate type
int x: parameter

Collection → Kumpulan / Koleksi dari object / instance

- Single

? Array.Resize

① Array

- Fix size
- Index
- Type-safety

- int[]

namaArray = new int[4];

= {1, 2, 3, 4};

- .NET 8 = [1, 2, 3, 4];

=

② ArrayList

- Dynamic
- not type-safety
- index

```
ArrayList myArray = new();
myArray.Add(3);
myArray.Add(505);
myArray.Add(true);
myArray.Add(505.2f);
```

```
myArray[0]
    ↓
int result = (int)
```

③ List<T> → System.Collections.Generic

+ Dynamic
- Index
- Type Safety

```
List<int> numbers = new();
numbers.Add(3);
numbers.Add(4);
```

numbers[0]

3

4 HashSet<T>

Dynamic
Typesafety
Index
Unique

```
HashSet<int> myhs = new();
myhs.Add(3);
myhs.Add(4);
myhs.Add(3);
```

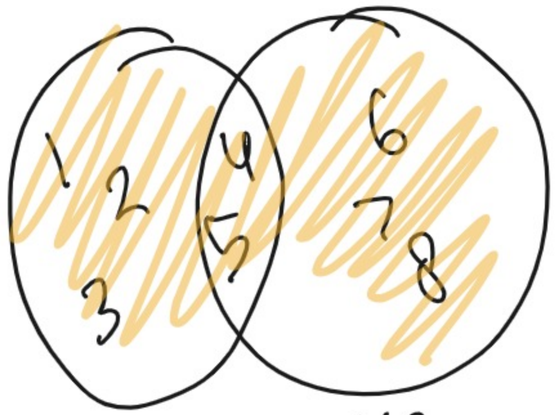
HashSet

- UnionWith
- IntersectWith
- ExceptWith

HashSet A(HA) → {1,2,3,4,5}

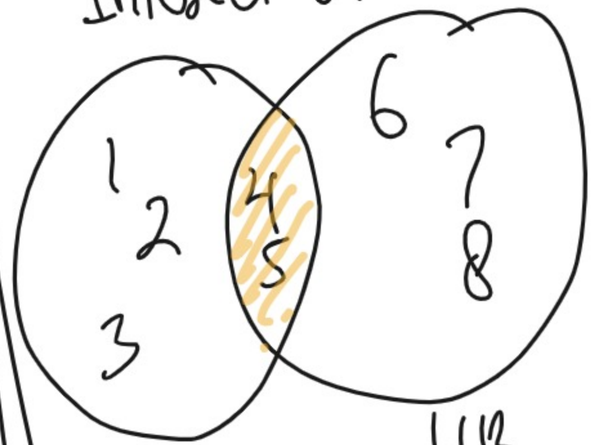
HashSet B(HB) → {4,5,6,7,8}

UnionWith



HR = {1,2,3,4,5,6,7,8}

IntersectWith



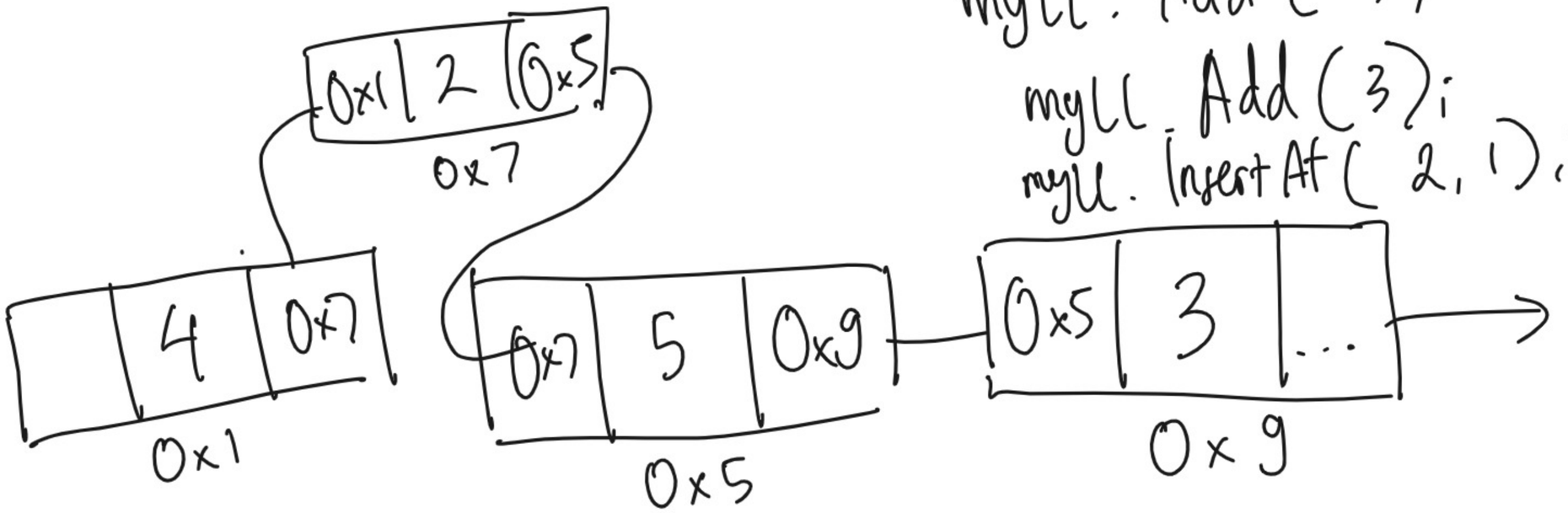
HR = {4, 5}

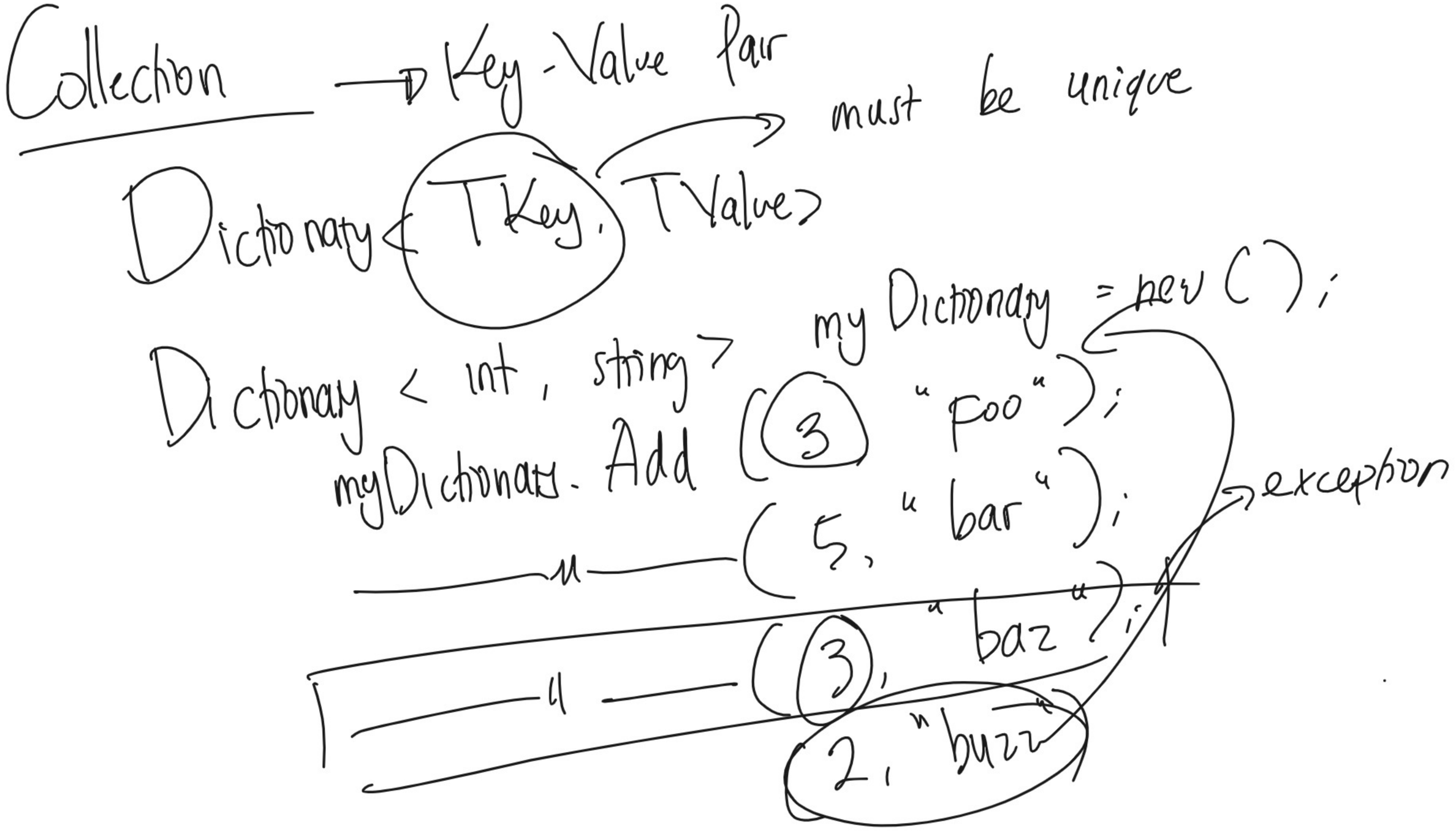
ExceptWith

HR = {1,2,3}

⑤ LinkedList<T>

```
LinkedList<int> myLL = new();
myLL.Add(4);
myLL.Add(5);
myLL.Add(3);
myLL.InsertAt(2,1);
```




Collection
Key-Value Pair
must be unique
Dictionary<TKey, TValue>
Dictionary<int, string>
myDictionary = new();
myDictionary.Add(3, "foo");
(5, "bar");
exception
(3, "baz");
2, "buzz"

myDictionary[TKey]

myDictionary[3]

SortedDictionary

SortedList

Dictionary yg sorted

KeyValuePair<T,T2>

Collection Single

Stack

Stack<T>

Push
Pop
Peek

LIFO = Last In First Out

```
Stack<int> myStack = new();
mystack.Push(3);
mystack.Push(5);
mystack.Pop(); → 5
mystack.Peek(); → 3
mystack.Peek(); → 3
```

Queue

Queue<T>

Enqueue
Dequeue
Peek

FIFO = First In First Out

```
Queue<int> myqueue = new();
myqueue.Enqueue(3);
myqueue.Enqueue(5);
myqueue.Dequeue(); → 3
myqueue.Peek(); → 5
myqueue.Peek(); → 5
```