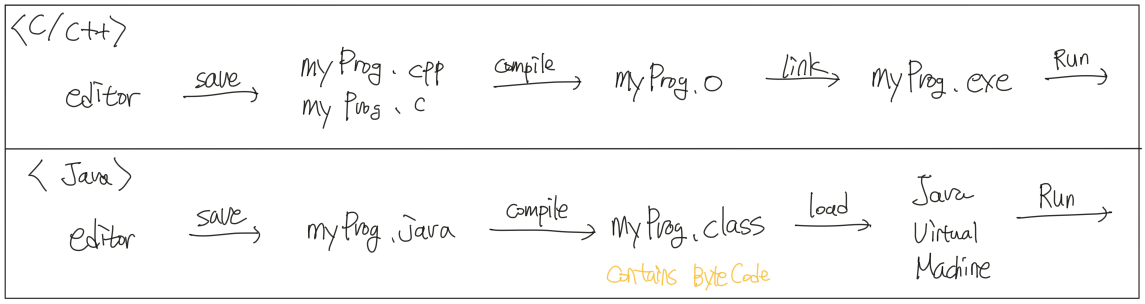
History of Programming Language

프로그래밍 언어	개발년도
BCPL	1967
B	1970
C (Not OOP)	1972
C++ (Combine OOP and non-OOP : Mixed Style, Hard to understand)	1995
Java (Only OOP : Everything in a Class, Consistent Style, Easy to understand)	1995



C/C++ Programs are compiled to Machine Instructions which only work on one Microprocessor.

Java Programs are Compiled to Byte-Code which can work on any Microprocessor.

The Java Virtual Machine (JVM) can quickly translate Byte-Code to machine Instructions and run those instructions.

The JVM uses a Just In Time (JIT) compiler. Every Microprocessor has its own JIT Compiler.

C/C++ files are compiled to object files. If one file uses code from another file, it must include the header of the other file.

Java files are compiled to class files. Every Java files has at least one class. The class name must be the same as the filename.

If a Java file uses code from a different file, then we only need to five the class name. The filename is the same, so the compiler can find that file.

In C++, `Circle c1` creates an instance of a `Circle`

In Java, `Circle c1` does not create an instance.

In Java, the only way to create an instance is with `new`

```
Circle c1;
c1 = new Circle();      //new creates an instance.
```

In C++ a constructor Initializes variables and might allocate memory.

If memory is allocated then a destructor is needed to deallocate that memory.

Java has constructors but not not destructors.

Java uses a Garbage Collector to deallocate memory, so destructors are not needed.

In Java, `c1 = new Circle();` `new Circle()` allocates memory and `c1` is a pointer to this memory.

This is sometimes called a "**Reference**" instead of a pointer.

Garbage Collector

If there is no longer any pointer to the allocated memory, then the Garbage Collector can deallocate that memory at any time.

In C++, we can say `Circle *c1;` `c1 = new Circle;`

In Java, we can say `Circle c1;` `c1 = new Circle();` (need ())

`c1` is a pointer.

In Java, no variable is an instance of a class, they can only be pointers.

In Java all variables are either Native Types and Reference Types.

Native Type : `int num;`

Reference Type : `Square sq1;`

Reference Types are pointers to a class instance.

There are 8 Native types

Type	Size (Byte)
boolean	1
char	2
byte	1
short	2
int	4
long	8
float	4
double	8

Assignment between Native Types has some restrictions.

`Integers` (byte, short, int, long) can be directly assigned to any `Floating Point Type` (float, double)

Within Integers or Floating Pointer Types smaller sized Types can be directly assigned to larger sized types.

For example)

```
int num = 3;
float value = 4.5;

value = num;        //ok
num = value;        //error
```

```
short x = 5;
int y = 7;

y = x;      //ok
x = y;      //error
```

```
float distance = 35.2;
double volume = 87.22;

volume = distance;      //ok
distance = volume;      //error
```

If you really want to assign in the wrong direction, the a cast must be used.

```
num = (int)value;
distance = (float)volume;
```

Write a program to calculate the area of a circle

```
//CircleArea.java
//Calculate the Area of a circle

import java.util.Scanner;   //Scanner is a class that allows input from a keyboard

public class CircleArea{
    public static void main(string[] args){
        Scanner input;
        double radius, area;
        double PI = 3.141592;

        input.new Scanner(System.in);
        System.out.print("Enter the Radius: ");
        radius = input.nextDouble();
        area = radius * radius * PI;
        System.out.printf("The area is %f\n", area);
    }
}
```

`System.out.print()` will print a message and stay on the same line.

`System.out.println()` will print a message and goto the next line.

`System.out.printf()` works like the C `printf` statement.

In Java it is a compiler error to read from a variable which is not yet assigned.

```
int x;
int y = 5;
y = x;      //error
```

Java does not even allow the possibility of an unassigned variable

```
int x;
int y = 5;
if(```) x = 7;
y = x;      //error
```

```
int x;
int y = 5;
if(y<4) x = 3;
if(y>=4) x = 7;
y = x;      //error
```

```
int x;
int y = 5;
if(y<4) x = 3;
else x = 7;
y = x;      //ok
```

Exception Handling

```
int x;
try{
    `
    `
    x = 0;
    `
    `
}catch(Exception ex){
    `
    `
    `
}
System.out.printf("%d", x);
```

If the exception occurs before we assign `x`, then the `COMPILE ERROR` might occur

```
int x;
try{
    `
    `
    x = 0;
    `
    `
}catch(Exception ex){
    `
    `
    x = 0;
    `
}
System.out.printf("%d", x);
```

It can be OK because there both have assigning `x`.

`throw new ArithmeticException()` Throwing an exception in `Java`. (Create an instance)

```
catch(Arithmetic Exception ex){         //ex is an instance of Exception.
    System.out.print(ex);
    throw ex;
}
```

```
void functionA(){
    try{
        ``` //possible exception
    }catch(Exception ex){
        System.out.print(ex);
        throw ex;
    }
    `
    `
    `
}

void funcitonB(){
    try{
        functionA();
    }catch(Exception ex){
        `
        `
        `   //Recover Here
    }
}
```

## Finally Block

```
try{
    `
    `
    `
}
catch(ArithmeticException ex){   //A
    `
    `
    `
}
catch(InputOutputException ex){   //B
    `
    `
    `
}
finally{   //C
    `
    `
    `
}
`   //D
`
```
```

```
23 `
```

- `A` instructions are performed only if there is an Arithmetic Exception inside `try`
- `B` instructions are performed only if there is an Input Output Exception inside `try`
- `C` instructions are will always be performed. (`catch`에 return이 있어도 무조건 실행됨)
- `D` instructions are performed if there is no exception or if an exception is caught.

Instructions in the `finally block` are performed if there is any exception or no exception.

One good use for a `finally block` is to close file which should be guaranteed to be closed;

Strings in Java

Strings are allocated in Memory, and cannot be changed in Memory after they are allocated.

However, String pointers can be changed to point to other strings.

```
String str1 = "abcde";
String str2 = new String("Hello");
String str3 = "abc" + "def";
char[] array1 = {"H", "i"};
String str4 = new String(array1);
//String constructor can accept string or array of characters
```

```
String s1 = "abc";
String s2 = "def";
String s3 = s1;
s1 = s1 + s2;
System.out.print(s1);   //"abcdef" (s1)이 가리키는곳이 새로운 곳으로 바뀐다.
System.out.print(s3);   //"abc"
```

`String Builder` is a different class of `string`.

An instance of the `String Builder` class can be modified in memory.

If a large String will frequently have small changed, then it may be better to use a String Builder instead of a String.

```
StringBuilder sb1 = new StringBuilder("abc");
StringBuilder sb2 = sb1;    //both points to same "abc" in memory.
sb1.append("def");      //cannot use opperator+
System.out.print(sb1);  //"abcdef"
System.out.print(sb2);  //"abcdef"
```

```
String str = new String("abc");
if(str == "abc")
//False because str points to a different location in memory than "abc"

if(str.equals("abc"))
//True because the "equals" method can compare the two strings.
```

```
String str = "abcdef";
str.substr(2); //"cdef"
str.substr(2,4); //"cd"
str.concat("ghi"); //"abcdefghi"
str.replace("d", "D");  //"abcDef"
```

All of these do not modify the original memory. All create a new string

An instance of any class can always be printed with the `System.out.print()`. This will use the method `toString` to convert the instance to a string

```
catch(Exception ex){
    System.out.print(ex);       //ex.toString()
}
StringBuilder sb1 = new StringBuilder("abc");
System.out.print(sb1);          //sb1.toString()
```

An instance of any class can use `toString`. This is because every class inherits from the `Object` class, and `Object` has a `toString` methods.

`toString` in the `Object` class just points some informations about the instance, such as size, name, locations.

Some classes override the `toString` method to print something more meaningful message.

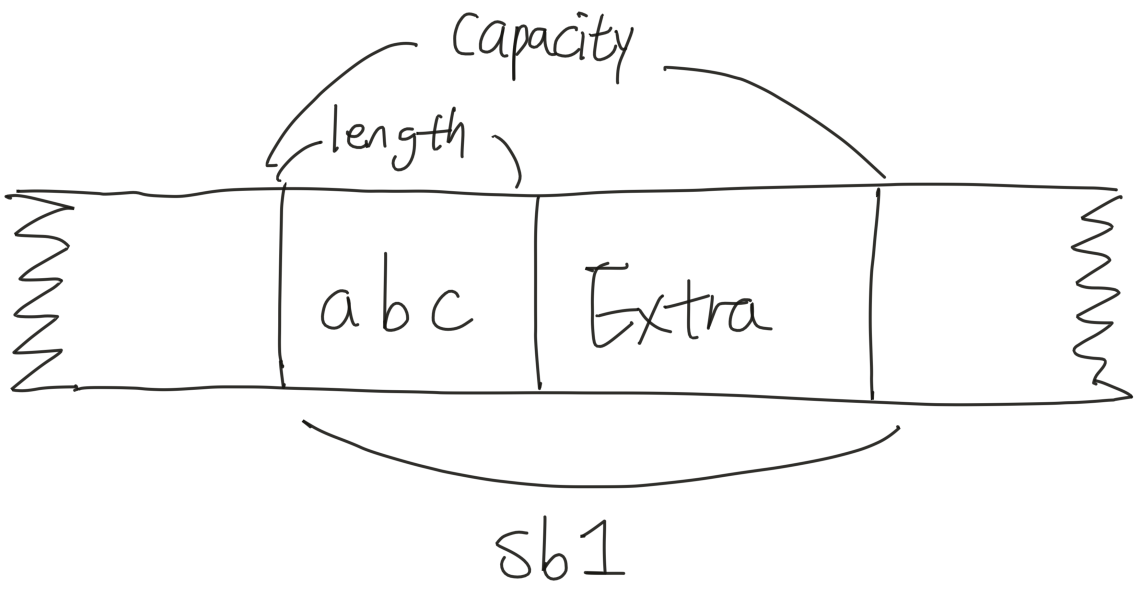
`StringBuilder` and `Exception` both override the `toString` method.

Any class you create will automatically have a `toString` method.

You can write your own `toString` method, and then `System.out.print()` will use your method instead of the `toString` method instead of the same method in `Object`.

If an instance of the `StringBuilder` class is created, then memory will be allocated for the `string`, plus some additional memory.

```
StringBuilder sb1 = new StringBuilder("abc");
```



`sb1.length()` is the number of characters in the string

`sb1.capacity()` is the size of `sb1` in memory.

Extra space is allocated so that the string can be made larger without taking too much time.

Arrays

```
import java.util.Arrays;
int[] myArray{4,3,7,6,8};

Arrays.sort(myArray);
```

The array can be changed but its size cannot be increased.

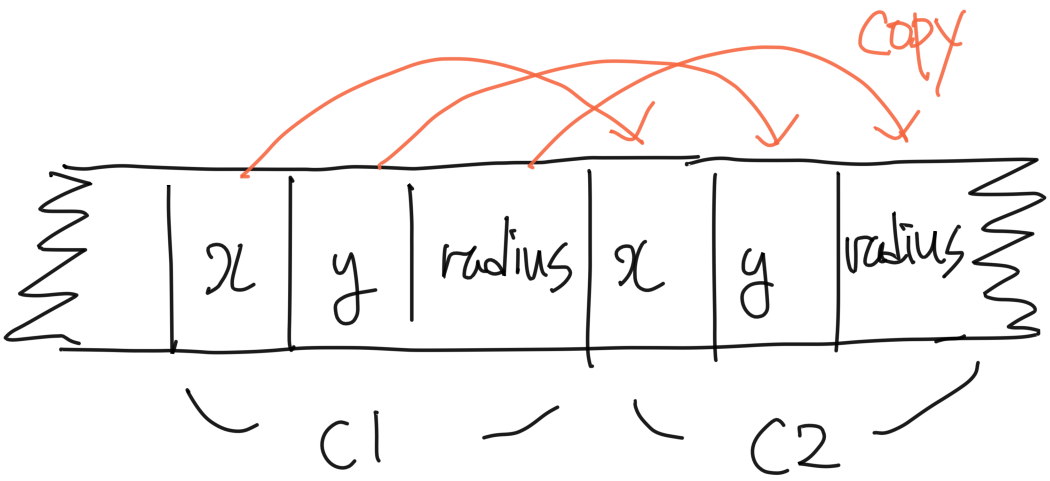
If we want an `Array` whose size can be increased, then we should use the `ArrayList` class instead.

```
import java.util.ArrayList;
ArrayList<Integer> numArray;

numArray.add(5);        //add the number 5 to the ArrayList
```

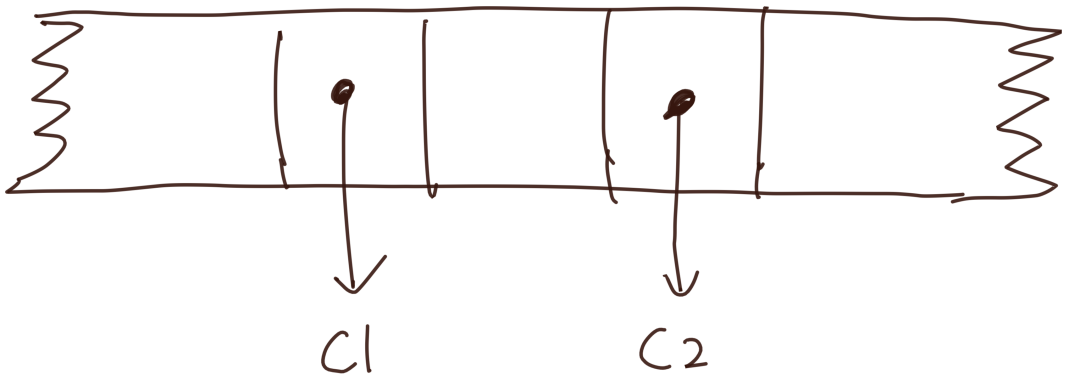
In `C++`

```
Circle c1, c2;
c2 = c1;
```

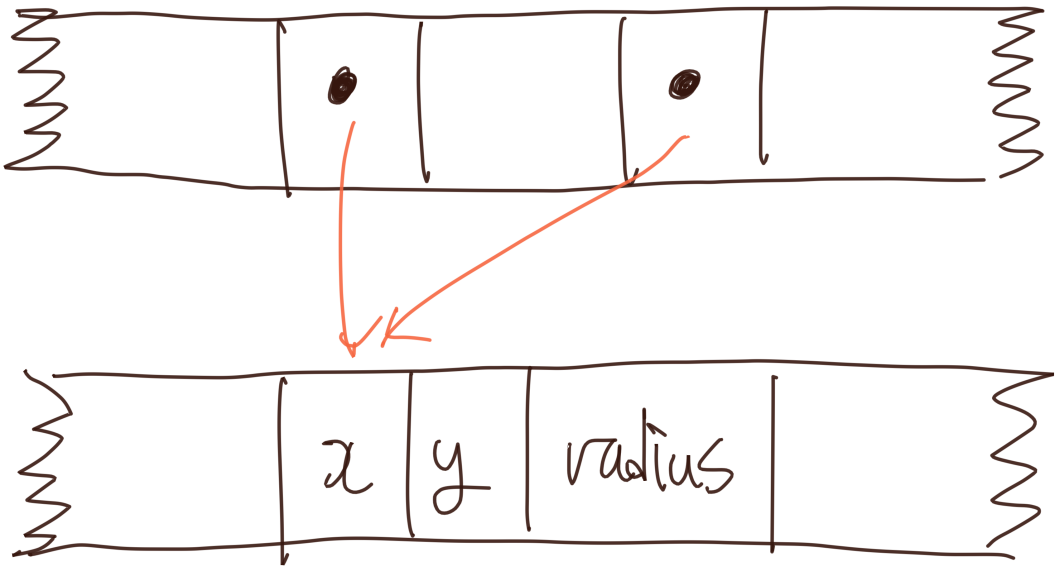


In `Java`

```
Circle c1, c2;
```




```
Circle c1, c2;
c1 = new Circle();
c2 = c1;
```



Inheritance in Java

```
public class Shape{
    private double xCenter;
    private double yCenter;
    public double getX(){
        return xCenter;
    }
    public double getY(){
        return yCenter;
    }
    public void setX(double newX){
        xCenter = newX;
    }
    public void setY(double newY){
        yCenter = newY;
    }
}

```

```
public class Circle extends Shape{  //Circle is a subclass of Shape in java
    private double radius;
    public double getRadius(){
        return radius;
    }
    public void setRadius(double newRadius){
        radius = newRadius;
    }
}
```

`C++` : Multiple Super classes possible.

`Java` : Only one super class.

	Class	Abstract Class	Interface
Can create an instance?	Yes	No	No
Can have methods?	Yes	Yes	No

An abstract class can have a list of abstract methods in Java. Any subclass of the abstract class must write all abstract methods or be an abstract class.

```
public abstract class Shape{
    private double xCenter, yCenter;
    //any subclass of Shape must have a getArea method.
    public abstract double getArea();
}
```

Abstract methods in `Java` are like pure virtual methods in `C++`. Any subclass of `Shape` must have a `getArea()` method or be an abstract class.

An interface is similar to an abstract class, but all methods in an interface must be abstract.

A class can inherit from many interfaces using "implements" but only one class or abstract class using "extends"

How to call a super class constructor from a sub class constructor?

`C++`

```
Circle::Circle(){
    Shape();        //call shape constructor
}
```

`java`

```
public class Circle{
    public Class(){
        super();    //call shape constructor (because of only one superclass)
    }
}
```

Recursion or Iteration

- Recursion

  Use a smaller version of the problem to solve a larger version of that same problem

- Iteration

  Solve the Problem using a loop that repeats the same steps

How to calculate Factorial problem?

0! = 1, 1! = 1, 2! = 2, 3! = 6, 4! = 24, ...

- Recursive Factorial

  `n! = n * (n-1)`

- Iterative Factorial

  `n! = 1 * 2 * ... * (n-1)`

```java
//Recursive Factorial
public int recFact(int n) throws Exception{
    if(n == 0)  return 1;
    if(n > 0) return n * recFact(n-1);
    if(n < 0) throw new Exception("n<0");
}

//Iterative Factorial
public int iterFact(int n) throws Exception{
    int c;
    int total = 1;
    if(n < 0) throw new Exception("n<0");
    if(n == 0) return 1;
    for(c = 1; c<=n; c++){
        total = total * c;
    }
    return total;
}
```

Layouts

If a `JFrame` contains more than one component, a Layout should be used to manage their Locations.

Flow Layout

`그림`

A `Flow Layout` adds component from left to right at the top of the `JFrame` It starts a second row after the edge of the `JFrame` is reached.

If the Frame is resized then the components can changed their Locations.

Grid Layout

그림

A `Grid Layout` arranges the components in a regular grid.

If the Frame is resized, then the components will also be resized, but their location will not be changed

Border Layout

그림

A `Border Layout` divides the Frame into Regions : North, South, East, West and Center.

With Flow Layouts and Grid Layouts it is necessary to create an instance of the Layout class.

```
class MyFrame extends JFrame{
    public MyFrame(){
        super();
        setLayout(new FlowLayout());
        //setLayout(new GridLayout());
        add(new JBotton("Push me"));
    }
}
```

With Border Layout we do not need to create instance of the Border Layout

```
class MyFrame extends JFrame{
    public MyFrame(){
        super();
        add(new JBotton("Push me"), BorderLayout.NORTH);
        addListener(new MyListener());
    }
}

```

```
private class MyListener implements ActionListenr implements ItemListener
    //Listen for Buttons, Listen for CheckBoxes
```

`implements` is used with an Interfaces.

A class can implement multiple interfaces but then only extend one super class.

Interface

An `Interface` can force the class to include specific methods.

If a class implements `ActionListener`, then the class must include the `actionPerformed` method

If a class implements `ItemListener`, then the class must include the `itemStateChanged` method.

How to convert between Classes and Native types

from → to	String	StringBuilder	int	float
String	X	new Stringbuilder(§)	Integer.parseInt(§)	Float.parseFloat(§)
StringBuilder	§.toString()	X	Integer.parseInt(§.toString())	Float.parseFloat(§.toString())
int	String.format("%d", §)	new StringBuilder(String.format("%d", §))	X	=§
float	String.format("%f", §)	new StringBuilder(String.format("%f", §))	=(int)§	X

Generic class

similar to `C++` templates.

```
public class Box<T>{
    private T element;
    public Box(T newElem){
        element = newElem;
    }
    public T getElem(){
        return element;
    }
}

public class BoxTest{
    public static void main(String[] args){
        Clock c1 = new Clock(10, 30, 00);
        Box b1 = new Box(c1);       //temporary container
        Clock c2 = b1.getElem();
        //c1 == c2
    }
}
```

Generic Stack

```
public class Stack<T>{
    private ArrayList<T> elements;
    public Statck(int capacity){
        elements = new ArrayList<T>(capacity);
    }
    public Stack(){
        this(10);
    }
    public void push(T value){
        elements.add(value);
    }
    public T pop(){
```

```
        if(elements.isEmpty()){
            throw new Exception("empty stack");
        }
        return elements.remove(elements.size()-1)
    }
}
public class StackTest{
    public static void main(String[] args){
        Stack<Integer> intStack;
        intStack = new Stack();
        intStack.push(4);
        intStack.push(8);
        System.out.print(intStack.pop())
    }
}
```

Generic Linked List

```
public class Node<T>{
    private T data;
    private Node<T> next;
    public void setData(T newData){
        data = newData;
    }
    public T getData(){
        return data;
    }
    public void setNext(Node<T> newNext){
        next = newNext;
    }
    public Node<T> getNext(){
        return next;
    }
}

public class LinkedList<T>{
    private Node<T> head;
    public push(T value){
        Node<T> newNode = new Node();
        newNode.setData(value);
        newNode.setNext(head);
        head = newNo

            de;
    }
}
```