M3

Java Programs

Classpath

Control panel:
└ System
  └→ Advanced system settings
    └ System properties
      └→ Advanced
        └→ Environment var
          └→ System var → (or user variables)
            └→ New / Edit : PATH ⇒ C:\ programfile \java\jdk1.8.0-60\bin;

To inform OS the JVM / Javac ~~has to find the class files~~ in the specified location

∵ 'Javac' is in Jdk1.8 \ bin \ javac

its better to use system variables

Compile:

```
> Javac firstProgram.java
```

Execution.

```
> java firstProgram.
```

(or)

Cmd prompt

Windows

```
set path = C:\programfile\java\jdk1.8\bin.
```

Mac os

```
export JAVA_HOME = /Library/Java/Home
echo $ JAVA_HOME
```

Helloworld.java

```
public class Helloworld
{
    public static void main (String args[])
    {
        System.out.println ("Helloworld");
    }
}
```

I. Variables:

Object stores its state in fields (or variables)

Kinds of Variables: → instance [non-static] → class [static] → Local → Global not ~~[illegible]~~ supported but can be implemented using static & public

(i) instance variables: [non-static fields]

⇒ Objects are declared as ~~instance~~/ stored as non-static fields

⇒ Because their values are unique to each instance of a class

⇒ associated with the ~~class~~ object

⇒ declared btw curly braces of a class but outside any method ⇒ also known as class members

(X) ⇒ memory is allocated when an object is created ⇒ eg: color different balloon object has a color.

Note: Three packages are imported by default.

(i) Java compiler ⇒ java.lang.*;

(ii) class Variables (or static fields)

i.e. if the instance variable uses 'static' keyword then called as class variable (X)

⇒ This tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.

i.e. Same instance variable of the static; ⇒ associated with class ⇒ called as static because stay the same for all objects ⇒ use static keyword (X)

eg: static int count = 0;

(iii) local variable:

⇒ inside a method or block of code scope

⇒ Method store its temporary state in local variables

⇒ an object stores its state in fields.

eg: int count = 0;

⇒ no local keyword. Variables declared in open brace & close braces indicates local & not accessible from rest of the class

(iv) ⇒ Parameters:

(X) ⇒ Parameters are always variables not fields

Fields ⇒ instance & class (not local & parameters)

Variables ⇒ all the above or local & parameters especially

II Naming:

(i) variable names are case sensitive

(ii) use unicode letters & digits (UTF-16)

→ begins with letter, $, or underscore

→ Subsequent characters may be letters, digits or underscore or $

→ no keyword or reserved word used as variable name

→ One word variable
  ⇒ spell in lower case letters
     eg: gear
→ two word or more
  ⇒ capitalize the first letter of each subsequent word
  eg: currentGear.
→ constant variable:
  ⇒ capitalize each letter
  eg: final int ~~max~~ MAX = 50;
     (or)
     final int MAX_SUBJECT = 5;
  ↙
  capitalize even for subsequent words & use underscore.

Quiz:

1) which type of variable requires a keyword to determine its scope.
   [✓ (i) local] (ii) static (iii) instance (iv) none

ie) variable is in scope only with in its own method no other code in the class can see.
When we call second time, it recreate the local variable & reinitialize.
Local variables must be initialized.

{ Local variable is on the Stack
  Object reference heap }

⇒ Local variable cannot use instance variables modifiers such as
   public, transient, volatile, abstract, static
   But can use final.

Class variable
(ii) (i) Every instance of the class shares a class variable.
 (ii) Any object can change the value of a class variable.
 (iii) Can be manipulated without creating an instance of a class

Class variable Example:

```
~~public class ClassVariable {~~
    ~~p.s.v.m (String args[])~~
    ~~{~~
       ~~private static int count = 10;~~

Public class ClassVariable {
   private static int count = 10;

   Public static void main (String args[])
   {
     ClassVariable c1 = new ClassVariable();
     ClassVariable c2 = new ClassVariable();
     ClassVariable c3 = new ClassVariable();

     S.o.pn ("c1.count" + c1.count
       + c2.count + c3.count);

     c1.count = 50;

     S.o.pn ("c1.count" + c1.count +
     "c2.count" + c2.count + "c3.count"
     c3.count);

     S.o.pln ("class.count" + ClassVariable.
                 count);
   }
}
```

O/P:
c1.count 10 10 10
c1.count 50 c2.count 50 c3.count 50
class count 50.

```
Public class Jaya
{
   private static int count = 10;
}

Public class ClassVariable
{ p.s.v.m (String args[])
  {
     ~~class~~ Jaya j = new Jaya();
     j.count = 50;
     S.o.pn ("count" + j.count);
  }
}
```

error
cannot access
private outside
the class.

A static method cannot access a nonstatic (instance) variable, because there is no instance

```
eg:   Public class jaya
      { int count = 10;
        P.s.v.m (String args[])
        {
          S.o.pn ("count" + count);
        }
      }
```

non static
variable count
cannot be referenced
from a static method

∴
```
Public class jaya
{ static int count = 10;
    p.s.v.m (String args[])
    { s.o.pn ("count" + count);
    }
}
```

Static method cannot
directly invoke a
non static method

[ie static = class
  nonstatic = instance

instance Variable

✓ ⇒ associated with object

✓ ⇒ defined inside the class but outside of any method

⇒ initialized when the class is instantiated

⇒ values are unique to each instance of the class

⇒ lives as long as the object does

? If you define any data member as transient then it will not persistent

✓ Use four access levels
(i) Final (ii) transient (iii) private, public, protected & default

Cannot be
abstract, static (: if used static then become class variable

```
public class Person
{
    private int height = ~~[illegible]~~ ;   (default 0)

    public Person()
    {
    this.height = 0;
    }

    void setHeight ( int height )
    {
    this.height = height;
    }

    int getHeight () {
        return height;
    }
}

public class instanceVariable {
    p. s. v. m (String[] args)
    {
        Person Jaya = new Person();
        Person Surya = new Person();
        Jaya.setHeight(150);
        s.o.pn ("Height of jaya" + Jaya.getHeight());
        Surya.setHeight(200);
        s.o.pn ("Height of Surya" +
                    Surya.getHeight());
    }
}
```

Global Variable;

No direct Concept of global variable in java
but can implement using.

(i) static keyword & public access modifier

```
Public class GlobalVariable {

        Public Static int Max_Size = 100;
        public Static int Min_Size = 0;

}
```

Access it as from anywhere.

```
GlobalVariable. Max_size.
GlobalVariable. Min_size.
```

(or)

With the help of interface:

```
public interface GlobalVariable {
  // variables are implicitly public, static & final
     int Max_Size = 100;
     int Min_Size = 0;
}

public class Jaya implement GlobalVariable{
     p. s. v. m ( String args[] )
     {
       s. o. pln ( "maxsize" + Max_size);
}  }
```

Local Variables;

Can't use
⇒ access modifiers (public, private, protected)
⇒ static, abstract, volatile, transient

Can use
final

→ only within the method

```
Public class LocalVariable {

    p. s. v. m ( String[] args)
    {
       new LocalVariable(). read();
    }

  Public void read() {
       int Size = 10;
       S. o. pln ("size" + Size);
  }
}
```

Local variables must be initialized

```
int Size;
Size = Size+10;
```

X

```
int Size=10;
Size = Size+10;
```

✓