Java basic program template

```
/** Basic.java
 *  Explains:
 *  Statements
 *  Expression
 *  Block
 *  Comments
 *  Whitespaces
 */
public class Basic {   // start of block1
   public static void main(String args[]) { // start of block 2
      String text = "Hello, basic element"; // declare a variable with data type is
      string, this is also called as an expression (a statement that returning values)
      int size = 10; //this also called as an expression
      System.out.println(text);   // statement
      System.out.print(    size );   // statement, with white spaces
      System.out.println();
      int newSize = size + 1; //expression;
      System.out.printf("new size: %d", newSize);
      System.out.println();
   } //end of block 2
} // end of block 1
```

1. Statement

A programming *statement* performs a piece of programming action. It must be terminated by a semi-colon (;) (just like an English sentence is ended with a period) as in the example above.

2. Block

A *block* is a group of programming statements enclosed by braces { `System.out.println("Hello, world!")`}. This group of statements is treated as one single unit. There are two blocks in this program. One contains the *body* of the class Hello1. The other contains the *body* of the main() method. There is no need to put a semi-colon after the closing brace.

3. Comments

Comments are NOT executable statements and are ignored by the compiler.

But they provide useful explanation and documentation. *Use comments liberally*.

- A multi-line comment begins with /* and ends with */.
- An end-of-line comment begins with // and lasts till the end of the line.
- A multi-line comment begins with /** and ends with */ will be proceded by *javadoc*

```
$javadoc -d direktoryhasildoc namafile.java
```

4. Whitespaces

Blank, tab, and newline are collectively called *whitespace*. Extra whitespaces are ignored, i.e., only one whitespace is needed to separate the tokens.

But they could help you and your readers better understand your program.

Use extra whitespaces liberally.

5. Case Sensitivity

Java is *case sensitive* - a *Hello1* is NOT a *hello1*. The *filename* is also case-sensitive.

```
String name; String Name; // they are different
```

6. Identifier

An identifier is a name given to a package, **class**, interface, **method**, or **variable, constant**. It allows a programmer to refer to the item from other places in the program. To make the most out of the identifiers you choose make them meaningful and follow the standard Java naming conventions (http://java.about.com/od/javasyntax/a/nameconventions.htm). The background of naming convention is that a fact of different Java programmers can have different styles and approaches to the way they program. By using standard Java naming conventions they make their code easier to read for themselves and for other programmers. Readability of Java code is important because it means less time is spent trying to figure out what the code does, leaving more time to fix or modify.

for the time being, we will only focus on class, method, variable and constant.

The rules:

- reserved words cannot be used

abstract	assert	boolean	break	byte	case
catch	char	class	const*	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto*	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

  - they cannot start with a digit but digits can be used after the first character (e.g., name1, n2ame are valid
  - the only symbols you can use are the underscore (i.e., "_") and dollar sign (i.e., "$").
  - cant use operators example +, -, *, etc

Package	Class	Variable	Constant

Package its self represent a directory name that contain some class with similar context. For example if we want to store class files that handle mathematic function. We can then have package “match”.
 	
`package math;`
`class Calculator`
`public static void main(String[] argv){`
`  public double add(double opr1, double opr2){`
`  }`
`}`
	Identifier that represent a name of code block. With this name we can make a new object.
	
	`public class HelloKelas{`
	
	`  public static void main(String[] argv){`
	
	`  HelloKelas hc = new HelloKelas();`
	
	`  }`
	
	`}`
		Is a kind of identifier that can store a value that is can be changed while the program is running. This is for example to handle user's input.
		
		`public class HelloVariable{`
		
		`  public static void main(String[] argv){`
		
		`  String name;`
		
		`  name = "Sukiyem";`
		
		`  name = "Sukijah";`
		
		`  System.out.println(name);`
		
		`  }`
		
		`}`
			This is fixed value, unchangeable identifier. It's important for example to store a constant to determine the value of unit of values.
			
			`public class HelloConstant{`
			
			`  public static void main(String[] argv){`
			
			`  final String name;`
			
			`  name = "Sukiyem";`
			
			`  name = "Sukijah";`
			
			`  System.out.println(name);`
			
			`  }`
			
			`}`
			
			`$java HelloConsta nt.java`
			
			`HelloConstant.java:9: error: variable name might already have been assigned`
			
			`  name = "Sukijah";`
			
			`  ^`
			
			`1 error`

```
/** void.java
* sep 30, 2015
* reserved word on class name raising error
*/

class void{
 public static void main(String[] args) {
  //class code goes here...
}}
```

```
/**
* Circle.java
* Sep 30, 2015
* Playing with identifiers
*/
class Circle{
    //<Type> <identifier> = <value>;;
    final double PHI = 3.4; // declare a constant
    private double radius = 0;  //declare a variable
    //<Type> <identifier>;

    private String name;
    public static void main(String args[]){
    //<classname> <identifier>=new <classname>() ;
    Circle circleku = new Circle(10); // declare an instance /object  of class
    circleku.calKeliling(); // call the method of object
    System.out.println("Luas: " + circleku.calLuas());  // call the method of object
  }

  public Circle(double radius){
     this.name = "Crop Circle"; // assignment on initialization constructor
     this.radius = radius;
     System.out.println("Name Circle: "+ this.name);
     System.out.println("Radius:  " + this.radius);
  }

   // method 'calKeliling' is an identifier
  private void calKeliling(){
     System.out.println("Keliling: " + 2 * PHI * this.radius);
  }
  public double calLuas(){
    return  (PHI * this.radius * this.radius);
   }
}
```

```
/** Person1.java
* sep 30, 2015
* valid identifiers on variables, class name
*/


class Person1{ public static void main(String[] params)

  {

   String name = "Sunoto";
   String address1 = "Jl. Raya tegal – pemalang KM 123";
   String address2 = "Jl. Raya dimana saja 99";
   double weight = 65;
   System.out.printf("My name is %s, address1 %s, address2  %s and weight %.0f Kg", name,
address1, address2, weight);


   System.out.println();
   String _name = "Sunoto";
   String $address1 = "Jl. Raya tegal – pemalang KM 123";
   String __address2 = "Jl. Raya dimana saja 99";
   double _$_weight = 65;
   System.out.printf("My name is %s, address1 %s, address2  %s and weight %.0f Kg \n", _name,
$address1, __address2, _$_weight);
   }
}
```

```
/** Person2.java
* Sep 30, 2015
* Invalid identifiers
*/

class Person2{
  public static void main(String[] params){
   String 4name = "Sunoto";
   String .address1 = "Jl. Raya tegal – pemalang KM 123";
   String -address2 = "Jl. Raya dimana saja 99";
   double :weight = 65;
   System.out.printf("My name is %s, address1 %s, address2  %s and weight %.0f Kg", name,
address1, address2, weight);
   String finally = "no special notes;
   }
}
```

```
$ javac Person2.java

Person2.java:15: error: not a statement

  String 4name = "Sunoto";

  ^

Person2.java:15: error: ';' expected

  String 4name = "Sunoto";

     ^

Person2.java:21: error: not a statement

  double :weight = 65;

  ^

Person2.java:21: error: ';' expected

  double :weight = 65;
```

```
    ^
Person2.java:26: error: not a statement
  String finally = "no special notes;
  ^
Person2.java:26: error: ';' expected
  String finally = "no special notes;
     ^
Person2.java:26: error: 'finally' without 'try'
  String finally = "no special notes;
      ^
Person2.java:26: error: illegal start of expression
  String finally = "no special notes;
         ^
Person2.java:26: error: unclosed string literal
  String finally = "no special notes;
           ^
9 errors
```

Task

1. Make your own class file and compile, run it that contains 5 variables as the valid identifiers
2. Make your own class file that raise an error complaining invalid identifier on the class name
3. Fix the following codes

```
class true{

public static void main(String[] arg){
   int age = 20;
   int _#money = 9000000;
   String @tempat_terakhir_update = "";
   String #status_updateterakhir_facebook_sebelum_mengurung_diri_selama_1bulan_penuh_gara2_skripsi
= "galau";
  System.out.print(age);
}}
```