File Edit View Help

File Edit View Help

Programming Review

File Edit View Help

Procedural Programming

Variables

- Store information
- Containers for Data
- Values can Change
- Declaring
  - Creates a Variable
  - Type
  - Identifier
- Spoken:
  - “Reserve space in memory for this type called this identifier”

Syntax

```
<<type>> <<identifier>>;
```

Example

```
double j;
```

Variables Types

- Type corresponds to bytes in memory
- Only use the type when declaring
- Programming Languages are either
  - Strongly Typed
  - Loosely Typed
- Primitive Types
- Object Types
  - Reference
  - Contents
- **Bold** are the most commonly used primitive data types in this course

Primitive Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	**4 bytes**	**Stores whole numbers from -2,147,483,648 to 2,147,483,647**
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	**8 bytes**	**Stores fractional numbers. Sufficient for storing 15 decimal digits**
boolean	**1 bit**	**Stores true or false values**
char	**2 bytes**	**Stores a single character/letter or ASCII values**

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
		28
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
		28
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
…	…	…

File Edit View Help

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
o	'\u0000'	40
…	…	…

Variables Types

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
o	'\u0000'	40
???	???	42
…	…	…

Variable Identifiers

- Gives the variable a “name”
- Uses the name to retrieve and store information
- Case sensitive
  - int test, int TEST, int tEsT would be 3 different identifiers
- “Camel Casing” common practice used for identifiers
- Identifiers cannot
  - Start with a Digit
  - Have Spaces
  - Match a reserved word
- Identifiers should avoid
  - Special Characters
  - Confusing names

Good Examples

```
int test01;
double largeValues;
boolean inClass;
```

Bad Examples

```
int 1Test;//Started with a digit
double big vals;//Used a space
boolean class;//Class is a reserved word
```

Assignment Operator

- The equals symbol “=” is the assignment operator
- Stores values found on the right hand side (RHS) of the operator into the identifier found on the left hand side (LHS)
- Assignments are valid if the type matches are is at least compatible
  - Primitive types can be stored in other primitive types as long the type’s byte amount is less than or equal to value being stored
  - Otherwise “type casting” is required
  - Type casting does not round it cuts off everything past the decimal point “.”
- Spoken:
  - “Store this value in this container”

Syntax

```
<<identifier>> = <<value>>;
```

Examples

```
i = 0;
j = 22.3;
o = ‘h’;
i = (int)j;//Type cast from double to int
//Value stored in “i” is 22
```

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = ‘h’;
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	22.3	32
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	22.3	32
o	'\u0000'	40
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	22.3	32
o	'h'	40
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	22.3	32
o	'h'	40
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	**22.3**	32
o	'h'	40
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	**22**	28
j	**22.3**	32
o	'h'	40
…	…	…

Assignment Operator

- Declare and assigning initial values
  - Good programming practice to assign initial values
  - Shortens two statements into one
  - Types are not still used after the declaration

Example

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	22	28
j	22.3	32
o	'h'	40
…	…	…

Constants

- Establishes a value that cannot change
- Great for avoiding “magic numbers”
- Good programming practice
  - Make the scope public
  - Make it static
  - Capitalize all characters in the identifier

Syntax

```
public static final <<type>> <<identifier>> = <<value>>;
```

Examples

```
public static final double PI = 3.14159;
public static final int BOARD_SIZE = 10;
```

Math Operators

- Performs computation and then assigns the results
- Order of Operations
- Basic Math Operations
  - Addition “+”
  - Subtraction “-”
  - Multiplication “*”
  - Division “/”
- Mod Operator “%”
  - Returns the remainder after division
  - Ex: 15 % 2 = 1

Syntax

```
<<identifier>> = <<value>> <<operator>> <<value>>;
```

Examples

```
//Variables
int value = 64 % i + 32;
//Constants
public static final double PI = 3.14159;
public static final double PI_SQ = PI*PI;
```

Math Operators

- Compute and Assign (C&A) Operators
  - Shorthand for applying some operator and value to a variable
  - Same as:
    - <<identifier>> = <<identifier>> <<operator>> <<value>>;
    - Ex: i = i+1; i+=1; i++; //Same statements
- Common Versions
  - "+=" – add and assign
  - "-=" – subtract and assign
  - "*=" – multiply and assign
  - "/=" – divide and assign
  - "%=" – mod and assign
- Special versions
  - "++" – Increase by 1
    - Same as "+= 1"
  - "--" – Decrease by 1
    - Same as "-=1"

Syntax

```
<<identifier>> <<C&A operator>> <<value>>;
```

Examples

```
i += 128; //If i = 32 now it is 160
j %= 2; //If j = 28.0 now it is 0.0
```

Basic Output

- System.out.println();
- Prints to the standard system output, the console

Syntax

```
System.out.println(<<value>>);
```

Examples

```
int i = 22;
System.out.println(i);
```

Basic Input

- Use Scanner to read from Console
- Must import type Scanner from “java.util” package
  - import java.util.Scanner;
- Create an instance of type Scanner that “scans” the standard system input
  - Scanner keyboard = new Scanner(System.in);
- Useful methods
  - next()
  - nextLine()
  - nextInt()
  - nextDouble()
- Also can be used to “scan” Strings, files, network traffic, etc.

Examples

```
Scanner keyboard = new Scanner(System.in);
String name = keyboard.nextLine();
int i = keyboard.nextInt();
keyboard.nextLine();//Useful “fix-up”
double j = keyboard.nextDouble();
keyboard.nextLine();//Useful “fix-up”
System.out.println(name+ “ “ + i + “ “ + j);
```

Console

```
JJ
64
3.14
JJ 64 3.14
```

Strings

- Object type
- Array of characters
- Denoted by double quotes (“”)
  - Characters are single quotes (‘)
- The plus (+) operator concatenates a value with a String
- Useful methods
  - charAt(index)
  - substring(startIndex)
  - substring(startIndex, endIndex)
  - toUpperCase()
  - toLowerCase()
  - split(regular expression)

Examples

```
String str = “abcdefg”;
System.out.println(str.charAt(0));
String str2 = str.substring(2,5);
System.out.println(str2);
```

Console

```
a
cde
```

Strings

- Object type
- Array of characters

Examples

```
String str = “abcd”;
```

Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = “abcd”;
```

Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = “abcd”;
```

Memory

Identifier	Contents	Byte Address
…	…	…
str	Null	28
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = “abcd”;
```

Memory

Identifier	Contents	Byte Address
…	…	…
str	Null	28
…	…	…
str[0]	‘\u0000’	64
str[1]	‘\u0000’	66
str[2]	‘\u0000’	68
str[3]	‘\u0000’	70
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = "abcd";
```

Memory

Identifier	Contents	Byte Address
…	…	…
str	Null	28
…	…	…
str[0]	'a'	64
str[1]	'b'	66
str[2]	'c'	68
str[3]	'd'	70
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = "abcd";
```

Memory

Identifier	Contents	Byte Address
…	…	…
str	**64**	28
…	…	…
str[0]	'a'	64
str[1]	'b'	66
str[2]	'c'	68
str[3]	'd'	70
…	…	…

Strings

- Object type
- Array of characters

Examples

```
String str = "abcd";
```

Memory

Identifier	Contents	Byte Address
…	…	…
str	**64**	28
…	…	…
str[0]	'a'	64
str[1]	'b'	66
str[2]	'c'	68
str[3]	'd'	70
…	…	…

Flow Control

- If-statement
- If the Boolean expression is “true” then the body of the if-statement is executed, and otherwise is ignored

Syntax

```
System.out.println(<<value>>);
```

Examples

```
int i = 22;
```

Branching Statements

- If-statement
- If the Boolean expression is “true” then the body of the if-statement is executed, and otherwise is ignored
- Putting curly braces “{}” to denote the body of the if-statement is strongly encouraged
- Do not put a semicolon “;” after the parenthesis
  - It will ignore the Boolean expression
- Spoken
  - “if this is true then do this”

Syntax

```
if(<<Boolean expression>>)
{
      //Body of the if-statement
}
```

Examples

```
if(a == b)
{
      System.out.println(“a is equal to b”);
}
```

Branching Statements

- Else-statement
- Requires a proceeding if-statement
  - If-statements do not require an else-statement
- If the Boolean expression is “false” then the body of the else-statement is executed, and otherwise is ignored
- Putting curly braces “{}” to denote the body of the else-statement is strongly encouraged
- Spoken:
  - “if this is true then do this, otherwise (else) do that”

Syntax

```
if(<<Boolean expression>>)
{
	//Body of the if-statement
}
else
{
	//Body of the else-statement
}
```

Examples

```
if(a == b)
{
	System.out.println(“a is equal to b”);
}
else
{
	System.out.println(“a is not equal to b”);
}
```

Branching Statements

- Else-If-statement
- Shorthand for an if-statement inside the body of an else-statement that connected to proceeding if-statement
- Requires a proceeding if-statement or else-if-statement
- Putting curly braces “{}” to denote the body of the else-statement is strongly encouraged

Syntax

```
if(<<Boolean expression>>)
{
        //Body of the if-statement
}
else if(<<Boolean expression>>)
{
        //Body of the else-if-statement
}
```

Examples

```
if(a < b)
{
        System.out.println(“a is less than b”);
}
else if(a > b)
{
        System.out.println(“a is greater than b”);
}
else
{
        System.out.println(“a and b are equal);
}
```

File Edit View Help

Branching Statements

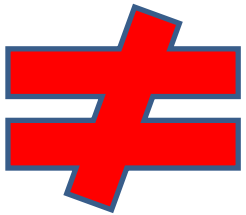
```
if (Boolean_Expression_1)
{
     Statement_1
}
else if (Boolean_Expression_2)
{
     Statement_2
}
else if (Boolean_Expression_3)
{
   Statement_3
}
else
{
     Default_Statement
}
```

```
if (Boolean_Expression_1)
{
     Statement_1
}
else
{
   if (Boolean_Expression_2)
   {
         Statement_2
   }
   else
   {
        if(Boolean_Exp_3)
        {
            Statement_3
        }
        else
         {
              Default_Statement
        }
   }
}
```

File Edit View Help

Branching Statements

```
if (Boolean_Expression_1)
{
     Statement_1
}
else if (Boolean_Expression_2)
{
     Statement_2
}
else if (Boolean_Expression_3)
{
   Statement_3
}
else
{
     Default_Statement
}
```



```
if (Boolean_Expression_1)
{
     Statement_1
}
if (Boolean_Expression_2)
{
     Statement_2
}
if (Boolean_Expression_3)
{
   Statement_3
}
else
{
     Default_Statement
}
```

Boolean Expressions

- True or False Value
- Common Boolean Operators
  - “==“ : Equal to
  - “!=“ : Not Equal
  - “<“ : strictly less than
  - “>” : strictly greater than
  - “<=“: less than or equal to
  - “>=“: greater than or equal to

Syntax

```
<<value>> <<Boolean operator>> <<value>>;
```

Examples

```
boolean a = 12 > 3;
if(a)//Or a == true
{
        System.out.println(“Here”);
}
else
{
        System.out.println(“Not here”);
}
```

Compound Boolean Expressions

- Combines multiple Boolean expressions
- Common Compound Boolean Expression Operators
  - "&&" : AND – both must be true to yield true
  - "||" : OR – only one must be true to yield true
  - "!" : NOT – negates the value. Not a binary operator like AND or OR
- Truth Table

A	B	A && B	A || B
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

Syntax

```
<<Boolean expression>> <<operator>> <<Boolean expression>>;
```

Examples

```
boolean a = 2 != 0 && 12 > 3;
if(a)//Or a == true
{
        System.out.println("Here");
}
else
{
        System.out.println("Not here");
}
```

Loops

- Runs a block of code some number of times
- The body of the loop runs while a Boolean expression is true
- While-loops are “check then iterate”
  - The body of the loop may run 0 to many times
- Great when unsure how many times the loop needs to run
- Spoken:
  - “While this is true, keep doing this”
  - “Until this is false, keep doing this”

Syntax

```
while(<<Boolean Expression>>)
{
      <<Body of the Loop>>
}
```

Examples

```
while(!gameOver)
{
      gameLoop();
}
```

Loops

- Do-While-loops are “iterate then check”
  - The body of the loop may run 1 to many times
- Great when unsure how many times the loop needs to run, but also needs to run the body of the loop at least once.
- Semi-colon must come after the while or it’s a syntax error
- Spoken:
  - “Do this While this is true”

Syntax

```
do
{
      <<Body of the Loop>>
}while(<<Boolean Expression>>);
```

Examples

```
do
{
     eat();
}while(full == false);
```

Loops

- For-loops are “counting loops”
  - The body of the loop runs some number of times.
- Great when it is known how many times the loop must run
- The sequence of a for-loop goes as
  1. Initialize counter
  2. Check Boolean Expression
  3. Run Body of the Loop
  4. Update the counter and go back to Step 2.
- Spoken:
  - “For this starts here and ends here, do this that many times”

Syntax

```
for(<<initialize counter>>;<<Boolean expression>>;<<update counter>>)
{
        <<Body of the Loop>>
}
```

Examples

```
for(int i=0;i<10;i++)
{
     System.out.println(i);
}
```

Arrays

- A collection (data structure) of items of the same type
- Fixed, contiguous block in memory
- Cannot resize in memory
  - Size of the array needs to be known before it is created
- Java arrays are considered “Objects”
  - Separated reference and contents
  - Has built in properties like “.length”
- When arrays are constructed all items are assumed to be assigned default values, in Java
- Indices (singular “index”) is how we can access and modify values in an array
- Valid indices start from 0 until Length – 1
  - If an array had 10 elements then the valid indices are from 0 to 9
- Array’s “best friend” is a for-loop
  - The loop can index into the array using its counter

Syntax

```
//Declaring and Constructing an Array
<<type>>[] <<identifier>> = new <<type>>[<<size>>];
//Indexing into an array to access a value
<<identifier>>[<<index>>];
//Indexing into an array to assign / modify a value
<<identifier>>[<<index>>] = <<value>>;
```

Examples

```
int[] i = new int[5];
i[2] = 22;
System.out.println(i[2]);
```

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	NULL	28
…	…	…

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	NULL	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	0	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	0	123

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	0	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

//Memory Address = 4 * 0 + 72

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	0	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

//Memory Address = 4 * 0 + 72

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	0	123

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	1	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	1	123

File Edit View Help

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	1	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

//Memory Address = 4 * 1 + 72

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	0	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	1	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

//Memory Address = 4 * 1 + 72

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	**2**	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	1	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

//Memory Address = size of the type * index + start Address

//Memory Address = 4 * 1 + 72

Memory

Identifier	Contents	Byte Address
…	…	…
a	**72**	28
…	…	…
a[0]	**0**	72
a[1]	**2**	76
a[2]	0	80
a[3]	0	84
a[4]	0	88
…	…	…
i	2	123

Arrays

Examples

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
    a[i] = i*2;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
a	72	28
…	…	…
a[0]	0	72
a[1]	2	76
a[2]	4	80
a[3]	6	84
a[4]	8	88
…	…	…
i	5	123