

Identifiers

What?

- memory location in RAM
- other programs, called 'variables'

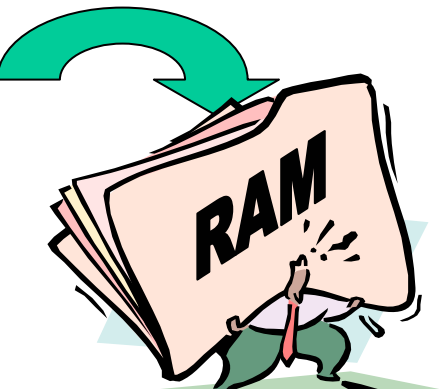


- 'declaring' is giving the memory a name

age name

- 'initializing' is giving the name's location a value

age name



Identifiers

Where Declared?

- to start, declare in the 'main' block

... program header

```
static public void main (String args[])
```

```
{ int age; // primitive identifier
```

```
    String name; // object identifier
```

... rest of program



- in the future, declared
 - a) in the class (global)
 - b) in the method (local)
 - c) in main (global/local)

Identifiers

Where Initialized?

```
int age;
```

```
String name;
```

```
age = 16;
```

```
String name = new String ("Mickey Mouse");
```

- as soon in the program as possible
- identifiers are NOT always assumed to be 0 (zero) or Null (empty)!
(error will be: identifier not assigned a value)
- identifiers CAN be filled (have data assigned) by user input statements (user keyboard entry)
☺*coming soon to a lesson near you*☺



'Primitive' Identifiers

Identifier 'Type'	Description	Bytes of Memory
int	integer – holds non-decimal numbers (both negative & positive)	4
double	- holds decimal numbers (both negative & positive)	8
char	character – holds 1 letter only Use single quotes “ For example: key = 'a';	2
boolean	-holds the word true or false	1 bit

'Primitive' Identifiers

```
// identifier initialization
```

```
payout = 0;
```

```
// Introduction
```

```
c.clear ();
```

```
c.println ( "Dice Throwing Game");
```

```
//Dice Roll using a random number generation
```

```
dice1 = (int) (Math.random () * 6) + 1;
```

```
dice2 = (int) (Math.random () * 6) + 1;
```

```
// picks from 6 numbers, starting at 1 ie 1-6
```

```
// determining if the rolls match
```

```
if (dice1 == dice2)
```

```
    right = true;
```

'Primitive' Identifiers

```
// giving output message and calculating payout;
c.setCursor (5, 1);
c.println ( "You rolled " + dice1 + " and a " + dice2 );
c.setCursor (9, 10);
if (right == true)           // selection statement using the boolean
{
    c.println ( "A Pair!");
    payout = 100.00;
}
else if (Math.abs (dice1 - dice2) == 1) // absolute difference (no -'ve)
{
    c.println ( "You have a run");
    payout = 50.00;
}
else                               // default selection
{
    c.println ( "No Points");
}
```

'Primitive' Identifiers

```
c.print ( "Your payout is $");
c.println (payout, 5, 2);
c.setCursor(15,20);
c.println( "Press any key" );
key = c.readChar( );
if ( key == 'b' ) // characters use single quotes
    c.println ( "Bonus of $10" );
}
```

'Primitive' Identifiers

The ALU and RAM functions with primitive types are as you would expect:

Arithmetic:

```
total = valA + valB;
```

```
average = total / count;
```

Logic:

```
if (valA > valB)
```

```
    println(valA + " is greater than " + valB);
```

RAM storage:

```
price = 62.50; // = is an ASSIGNMENT command for RAM
```

'Primitive' Identifiers - Summary

- **'identifier'** is a name to identify a RAM memory location and its size (by virtue of the 'type' of identifier)
- identifiers are **declared** (create the memory location) and **initialized** (fills the memory location with a value)
- primitive data types always use lower case letters

'Object' Identifiers

- Computers do not really process words as we are led to believe they do.
- Programming languages are written to take care of the details for us.
- Words are called **Strings**
- **Strings** are really a collection of char (characters ... primitive type) which end with a 'Null String' (\0 is the symbol for null string which indicates the end of the word).
- For example, the **String** 'hello' is stored as h~e~l~l~o~\0, which is 6 characters!

'Object' Identifiers

- Declaration of words:

The type is called: **String** (notice the capital S)

For example:

```
String name;           // declaration alone  
String address = "123 Fourth Street"; // declaration with  
                                     // initialization combined
```

- Because Strings are not primitive types, but are object types, the commands to work with Strings also differ
- Note: The capital S in String is your hint that things are different!

'Object' Identifiers

The ALU and RAM functions with String Object types different than you would expect:

Concatination (joining by using the +):

```
fullName = firstName + " " + lastName);
```

```
fullName = firstName.concat(lastName);
```

RAM Assignment (initialization outside declaration line):

```
name = new String ( "Peter Pan" );
```

'Object' Identifiers

```
wor = new String ( "Peter Pan" );
```

P is in position 0, e is in position 1, t is in position 2, ...
- the position of the letters start at 0

```
c.println ( wor.substring(5) );
```

> r Pan ... gives last 5 characters

```
c.println( wor.substring(0,4))
```

> Peter (letters in positions 0 through 4)

```
x = wor.length( );
```

> x holds 9, the # of characters (which count from 0 to 8)

```
alet = wor.charAt (0);
```

> alet holds the character P

'Object' Identifiers

```
newWor = wor.replace ('a', 'i' );
```

> newWor now holds Peter Pin

```
c.println (wor.toLowerCase( ) );
```

> peter pan (takes away the capitals)

```
c.println (wor.toUpperCase( ) );
```

> PETER PAN (gives all capitals)

```
if (wor.equals ("peter pan") )
```

```
    c.println("Do not match");
```

> capitals are different, thus they are unequal

```
if (wor.equalsIgnoreCase ("peter pan") )
```

```
    c.println("Do match");
```

> equal, since the upper/lower case is ignored

'Object' Identifiers - Summary

- **String** is an object identifier, and has different 'rules' / 'ways' for its use
- **String** can not be directly compared (can't use =), but rather requires a String method (s1.equals(s2);)
- **String** can be manipulated with a number of String methods (such as .length, .substring)
- A clue to remember that **String** is different is the capital S in String in declaration!

Never-Changing Identifiers

- Identifiers that never change are called **CONSTANTS**.
- Constants are used to protect an identifier's value
- For example, now PST is 8%. If it changes to 6% in the future, we don't want to search through all programs, all lines, to make the necessary changes. We want to make the change once and that one change be effective for the entire program.
- Constants are identified by the word 'final' prior to the type, and the identifier is in all capitals: `final double GST = 0.08;`
- Constants are 'expanded' in the compiling stage of creating the bytecode. This means that all occurrences of the identifier are replaced by the value, and the identifier no longer exists.

Never-Changing Identifiers

// example

```
public class Example
{
    static Console c = new Console();
    public static void main (String args[])
    {
        final double GST = 0.07;
        final double PST = 0.08;
        double price, total;
        price = 10.00;
        total = (price * GST) + (price * PST) + price;
        c.print ( "Purchase: " );
        c.println(price,5,2);
        c.print ( "Total: " );
        c.println(total,5,2);
    }
}
```