

# **Variables**

# **Data Types**

**Lab 0B**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# What is a reference?

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# What is a reference?

A reference variable stores the memory address of an object.

```
Monster fred = new Monster();  
Monster sally = new Monster();
```



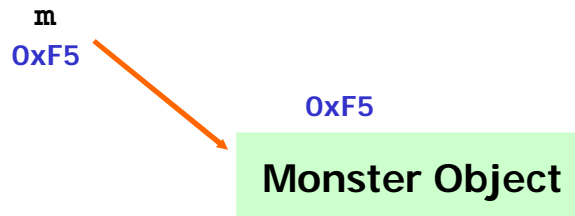
© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

A reference variable is used to store the location of an Object. In most situations, a reference stores the actual memory address of an Object.

`fred` and `sally` store the location / memory address of two new Monster Objects.

# What is a reference?

```
Monster m = new Monster();
```



`m` stores the address of a `Monster`

© A+ Computer Science - www.apluscompsci.com

A reference variable is used to store the location of an Object. In most situations, a reference stores the actual memory address of an Object.

`m` stores the location / memory address of a new `Monster`.

# What is a variable?

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# What is a variable?

A variable is a storage location for a specified type of value.

```
int numDays = 365;  
double hTownTax = 8.25;  
char grade = 'A';
```

numDays

365

hTownTax

8.25

© A+ Computer Science - www.apluscompsci.com

A non-reference variable is a storage location for a value.

numDays is an integer primitive variable. numDays is not a reference variable. numDays stores an integer value.

hTownTax is a double primitive variable. hTownTax is not a reference variable. hTownTax stores a decimal value.

# What is a variable?

```
int numDays = 365;
```

numDays

365

numDays stores an integer value

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

A variable is a box that stores a specific type of value. numDays stores an integer value. numDays is not a reference; thus, it does not store a location / memory address.

# Identifier Names

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)



# What does identifier mean?

An identifier is used to identify something.

```
public class Triangle{ }
```

```
int width = 7;
```

Always start identifier names with letters.

© A+ Computer Science - www.apluscompsci.com

An identifier is used to identify something. Identifiers should begin with letters. Identifiers can contain symbols, letters, and numbers.

A box that will store integer numbers needs a name. The name should clearly identify what will be stored in the box. `width` clearly states that the box will contain the width of something.

`Triangle` is used to identify a class. The assumption is that the class will store information about Triangles.

# Identifiers

Which of these would be legal identifiers?

1stYear  
jump Up  
feet2Inches  
BigTriangle  
SpaceInvaders



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

1stYear is not legal. Identifiers cannot start with numbers.

jump up is not legal. Identifiers cannot contain spaces.

feet2Inches is legal.

BigTriangle is legal.

SpaceInvaders is legal.

Space\_Invaders is legal.

\_SpaceInvaders is legal, but not a suggested naming style.

# Identifier Names

Always use names that mean something.

```
double totalPay;  
class Triangle{ }
```

```
double a;           //very bad  
class B{}           //very bad
```

© A+ Computer Science - www.apluscompsci.com

Use identifier names that are clear and informative.

The name `totalPay` seems to indicate the variable will store the total pay amount for someone or something.

```
double nationalDebt;  
char firstLetterOfLastName;  
long buildingHeight;  
public class BlinkyBall{}  
public class BlackJack{ }
```

# What is a keyword?

Keywords are reserved words that the language uses for a specific purpose.

**int   double   return   void  
static   long   break   continue**

Keywords cannot be used as identifiers.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Keywords are words that have been assigned a special purpose in the language. Keywords cannot be used as identifier names.

# Spelling Counts

**SAM does not equal sam.  
Sam does not equal sam.  
Same does not equal sam.**

**Case is important as is spelling.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Java is case sensitive.

# Open identifiers.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Types of Variables

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Data Types

byte  
long

short  
float

int  
double

int whole  
double fraction



The **type** states how much and what kind of data the variable can store.

© A+ Computer Science - www.apluscompsci.com

When defining a variable, a data type must be provided. The data type describes what will be stored in the variable. A variable is a box where things will be stored. The data type states what kind of things can be placed in the box.

`int` can store non-decimal positive and negative numbers.

`double` can store decimal positive and negative numbers.



# All Data Types

data type	memory usage	min .. max
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	-2 billion to 2 billion
long	64 bits	-big to +big
float	32 bits	-big to +big
double	64 bits	-big to +big
char	16 bit unsigned	0 - 65535
reference	32 bits	n/a

It is important to know all data types and what each one can store.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

This data type chart lists most data type's memory usage and range of storage values.

# Integers



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Integers

```
int one = 120;  
int two = 987123;  
byte bite = 99;  
long longInt = 99234423;
```

```
System.out.println(one);  
System.out.println(two);  
System.out.println(bite);  
System.out.println(longInt);
```

## OUTPUT

```
120  
987123  
99  
99234423
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Integer types(byte, short, int, long, and char) can only store non-decimal values.

# Integers

```
int one = 120.0;
```

```
System.out.println(one);
```

**OUTPUT**

LOP error

Integer types can store integer values only.  
Integer types cannot store fractional / decimal values.

Attempting to assign fractional / decimal values to an integer type results in a loss of precision compile error.

© A+ Computer Science - www.apluscompsci.com

`int one = 120.0;` results in an error. `120.0` is a decimal value and integer types cannot store decimal values.

`int one = (int)120.0;` type casting temporarily converts the receiving value so that it can be stored in an integer storage location.

**Open**  
**integers.java**  
**integerslop.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Real Numbers

## Fractional Values



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Reals

```
double one = 99.57;  
double two = 3217;  
float three = 23.32f;
```

```
System.out.println(one);  
System.out.println(two);  
System.out.println(three);
```

## OUTPUT

```
99.57  
3217.0  
23.32
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Real / decimal types (`float`, `double`) can store non-decimal values as well as decimal values.

```
double example = 456;  
example = 456.323;
```

# Reals

```
double one = 120.7;  
System.out.println(one);  
one = 125;  
System.out.println(one);
```

## OUTPUT

```
120.7  
125.0
```

Real types can store fractional/decimal values as well as integer values.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Real / decimal types (`float`, `double`) can store non-decimal values as well as decimal values.

```
double example = 456;  
example = 456.323;
```



# **Open reals.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Characters

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Characters

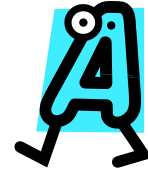
```
char let = 'A';  
char fun = 65;
```

```
char test = 'a';  
char go = 97;
```

```
char what = 48;
```

char variables are used to store a single letter.

char variables are actually integers.



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

char is an integer data type.

# Characters

char is a 16-bit unsigned int data type.

Here is a 16 bit pattern: 000000000110011

char let = 65;

## ASCII VALUES YOU MUST KNOW!

'A' – 65

'a' – 97

'0' - 48

© A+ Computer Science - www.apluscompsci.com

char is an unsigned(has no negative range) integer data type.

```
char letter = 97;  
out.println(letter);           //outs a  
letter = 'A';  
out.println(letter);           //outs A
```

# ASCII Values

'A' - 65    'B' - 66    'C' - 67    ...

'a' - 97    'b' - 98    'c' - 99    ...

'0' - 48    '1' - 49    '2' - 50    ...

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Once you memorize the starting value for 'A', 'a', and '0', determining the ASCII values for most letters and numbers is pretty simple.

# Characters

```
char alpha = 'A';  
char ascii = 65;  
char sum = 'B' + 1;
```

```
System.out.println(alpha);  
System.out.println(ascii);  
System.out.println(sum);  
System.out.println('B'+1);
```

## OUTPUT

```
A  
A  
C  
67
```

© A+ Computer Science - www.apluscompsci.com

Because char is an integer data type, it is okay to store non-decimal values in a char. It is also okay to perform integer math operations on a char variable and to store math results in a char variable.

```
char example = 98;  
out.println(example);           //outs a b  
  
example = 'A'+5;  
out.println(example);           //outs a F  
  
out.println('A'+5);             //outs a 70  
                                //outs a 70 because char + int nets an int
```

# Open chars.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Start work on Lab 0b

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)



# Booleans

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Booleans

```
boolean go = true;  
System.out.println(go);  
boolean stop = false;  
System.out.println(stop);
```

## OUTPUT

true  
false

A boolean type can store true or false only.



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

A boolean can store true or false. A boolean cannot store letters or numbers.

# Open booleans.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Strings

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Strings

```
String dude = "hello world";  
String buddy = "whoot - \\\\\\\\\\\\\\\\\\\\\\\\";  
  
System.out.println(dude);  
System.out.println("buddy = " + buddy);
```

## OUTPUT

```
hello world  
buddy = whoot - \\\\\\\\\\\
```

A String type stores groups of characters.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Open strings.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Variable Assignment

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# The Assignment Statement

```
receiver = 57;  
receiver = 239423;
```

In an assignment statement, the receiver is always on the left of the assignment operator ( = ).

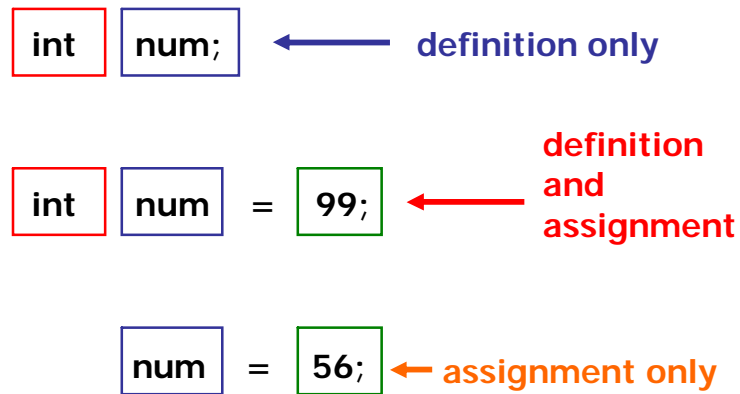
© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The variable receiving the value is placed on the left of the assignment operator( = ).

57 is the value being placed in box receiver.



# Defining VS. Assigning



© A+ Computer Science - www.apluscompsci.com

A data-type must be placed in front of/to the left of a variable name when that variable is defined.

When assigning a variable that has already been defined, only the name and the value are required.

# The Assignment Statement

```
int number = 75, bigNum=99;  
double hTownTax = 8.25;  
char bigA = 'A', littleA = 'a';  
boolean isPrime = false;  
String s = "abc";  
  
System.out.println(number);  
System.out.println(bigNum);  
System.out.printf("%.2f\n",hTownTax);  
System.out.println(bigA);  
System.out.println(littleA);  
System.out.println(isPrime);  
System.out.println(s);
```

## OUTPUT

```
75  
99  
8.25  
A  
a  
false  
abc
```

© A+ Computer Science - www.apluscompsci.com

Variable definitions and assignments can be performed on one line.

```
int intFun=75;           //definition and assignment
```

More than one variable can be defined and assigned on the same line.

```
int go=3, stop=2, pause=1; //separate with a comma
```

# Open assignment.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Data Type Ranges

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# All Data Types

data type	memory usage	min .. max
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	-2 billion to 2 billion
long	64 bits	-big to +big
float	32 bits	-big to +big
double	64 bits	-big to +big
char	16 bit unsigned	0 - 65535
reference	32 bits	n/a

It is important to know all data types and what each one can store.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

This data type chart lists most data type's memory usage and range of storage values.

# Memory

Memory consists of bits and bytes.

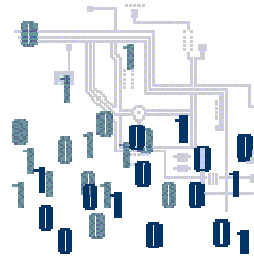


8 bits = 1001 0010 = 1 byte

16 bits = 0101 1001 0100 1001 = 2 bytes

The more bits you have the more you can store.

1 byte = 8 bits



© A+ Computer Science - www.apluscompsci.com

The more bits a data type has the more that data type can store. A 64 bit type has much more storage room than an 8 bit type.

128 64 32 16 8 4 2 1 base 10 value of each binary digit

1 0 1 0 = 10 in base 10

1 1 1 1 = 15 in base 10(4 bit)

1 0 0 0 1 0 0 0 = 136 in base 10

1 1 1 1 1 1 1 1 = 255 in base 10(8 bit)

# Integer MIN and MAX

```
System.out.println(Byte.MIN_VALUE);  
System.out.println(Byte.MAX_VALUE);
```

```
System.out.println(Short.MIN_VALUE);  
System.out.println(Short.MAX_VALUE);
```

MIN\_VALUE and  
MAX\_VALUE are  
very useful for  
contest  
programming.

## OUTPUT

```
-128  
127  
-32768  
32767
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The MIN\_VALUE and MAX\_VALUE fields store the minimum and maximum values that can be stored in a particular type.

# Integer MIN and MAX

```
System.out.println(Integer.MIN_VALUE);  
System.out.println(Integer.MAX_VALUE);
```

```
System.out.println(Long.MIN_VALUE);  
System.out.println(Long.MAX_VALUE);
```

## OUTPUT

```
-2147483648  
2147483647  
-9223372036854775808  
9223372036854775807
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The MIN\_VALUE and MAX\_VALUE fields store the minimum and maximum values that can be stored in a particular type.



# Overflow Errors

```
int num = Integer.MAX_VALUE;  
num=num+1;  
System.out.println(num);  
num=num-1;  
System.out.println(num);
```

Why does adding 1 to  
MAX\_VALUE give you the  
MIN\_VALUE?

**OUTPUT**  
-2147483648  
2147483647

© A+ Computer Science - www.apluscompsci.com

Overflow errors occur at run-time when a value is assigned to a variable that is too large. The resulting value is typically a negative value. The negative value occurs when the positive upper bound is overflowed into the negative range.

Attempting to assign a numeric constant that is too large to a variable is a syntax error. It is very easy for Java to determine that the value is too large for the data type.

```
byte example = 128; //compile error
```

# Open integersminmax.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Real MIN and MAX

```
System.out.println(Float.MIN_VALUE);  
System.out.println(Float.MAX_VALUE);
```

```
System.out.println(Double.MIN_VALUE);  
System.out.println(Double.MAX_VALUE);
```

MIN\_VALUE and  
MAX\_VALUE are  
very useful for  
contest  
programming.

## OUTPUT

```
1.4E-45  
3.4028235E38  
4.9E-324  
1.7976931348623157E308
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The MIN\_VALUE and MAX\_VALUE fields store the minimum and maximum values that can be stored in a particular type.

**Open**  
**realsminmax.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Character MIN and MAX

```
out.println((int)Character.MIN_VALUE);  
out.println((int)Character.MAX_VALUE);
```

```
out.println(Character.MIN_VALUE);  
out.println(Character.MAX_VALUE);
```

MIN\_VALUE and  
MAX\_VALUE are  
very useful for  
contest  
programming.

## OUTPUT

```
0  
65535  
?  
?
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The MIN\_VALUE and MAX\_VALUE fields store the minimum and maximum values that can be stored in a particular type.

**Open**  
**charsminmax.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Mixing data

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Mixing Data

Java is a strong typed language.  
You must pay attention to a variable's  
type when assigning a value.

```
int one=90;  
char letter= 'A';  
char let= 97;
```

```
one=letter;  
letter=let;  
one=let;
```



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

As Java is a strong-typed language, it is important to pay attention to data types when assigning values.

Integer types cannot store decimal values unless a cast is used.

Also, when assigning values from one variable to another, data types and data type sizes play an important part.



# Mixing Data

```
int one = 90;
double dec = 234;
char letter = 'A';

System.out.println( one );
one = letter;    //char to int
System.out.println( one );

one = 'A';      //char to int
System.out.println( one );

System.out.println( dec );
dec = one;      //int to double
System.out.println( dec );
```

## OUTPUT

```
90
65
65
234.0
65.0
```

Data type sizes  
often determine  
if assignment is  
legal.

32 bit == 32 bit

© A+ Computer Science - www.apluscompsci.com

When assigning values from one variable to another, data types and data type sizes play an important part.

Large size types can be assigned same size type and smaller type values.

Smaller size types cannot be assigned larger type values.

16 bit = 16 bit and smaller //legal

8 bit = 16 bit and smaller //illegal without a cast()

**open**  
**mixingdata.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Finish work on Lab 0b

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# AutoBoxing

# AutoUnboxing

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# References/Objects

In JAVA, you have 8 primitive data types.

All other variables in Java are reference variables. **References** refer to objects.

```
Monster m = new Monster();
```



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# References/Objects

primitive	object
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

For each of the primitive types, there is a corresponding wrapper class.

`int` has a corresponding wrapper class named `Integer`.

`int` stores a non-decimal value.

`Integer` stores the location / memory address of an `Integer` Object which stores an `int` value.

`double` has a corresponding wrapper class named `Double`.

`double` stores decimal and non-decimal values.

`Double` stores the location / memory address of a `Double` Object which stores a `double` value.

# Box/Unbox

Before Java 5 added in autoboxing and autounboxing, you had to manually wrap primitives.

```
Integer x = new Integer(98);  
int y = 56;  
x= new Integer(y);
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Before autoboxing and autounboxing, a reference could only refer to a primitive if a Wrapper class was instantiated and the primitive passed to the constructor.

# Box/Unbox

Java now wraps automatically.

```
Integer numOne = 99;  
Integer numTwo = new Integer(99);
```

```
=99;  
=new Integer(99);  
These two lines are equivalent.
```



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

`Integer numOne = 99;` is equivalent to  
`Integer numOne = new Integer(99);`

With the introduction of Java 5, the new `Integer()` Object instantiation code happens in the background. Java takes care of these details, but does not show the work.



# Box/Unbox

Java now wraps automatically.

```
Double numOne = 99.1;  
Double numTwo = new Double(99.1);
```

```
=99.1;  
=new Double(99.1);  
These two lines are equivalent.
```



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

`Double numOne = 99;` is equivalent to  
`Double numOne = new Double(99);`

With the introduction of Java 5, the new `Double()` Object instantiation code happens in the background. Java takes care of these details, but does not show the work.

# Box/Unbox

Before Java 5 added in autoboxing and autounboxing, you had to manually unwrap references.

```
Integer ref = new Integer(98);  
int y = ref.intValue();
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Before autoboxing and autounboxing, a reference value could only be stored in a primitive if the corresponding reference get method was called to retrieve the primitive value from the reference.

# Box/Unbox

Java now unwraps automatically.

```
Integer num = new Integer(3);  
int prim = num.intValue();  
out.println(prim);  
prim = num;  
out.println(prim);
```

```
prim=num.intValue();  
prim=num;
```

These two lines are equivalent.

**OUTPUT**

3  
3

© A+ Computer Science - www.apluscompsci.com

```
Integer numOne = new Integer(99);  
int primInt = numOne; is equivalent to  
int primInt = numOne.intValue();
```

With the introduction of Java 5, the `intValue()` method call happens in the background. Java takes care of these details, but does not show the work.

# Box/Unbox

```
Double dub = 9.3;  
double prim = dub;  
out.println(prim);
```

```
int num = 12;  
Integer big = num;  
out.println(big.compareTo(12));  
out.println(big.compareTo(17));  
out.println(big.compareTo(10));
```

## OUTPUT

```
9.3  
0  
-1  
1
```

© A+ Computer Science - www.apluscompsci.com

Before autoboxing and autounboxing, a reference could only refer to a primitive if a Wrapper class was instantiated and the primitive passed to the constructor.

Before autoboxing and autounboxing, a reference value could only be stored in a primitive if the corresponding reference get method was called to retrieve the primitive value from the reference.

With the introduction of Java 5, the wrapping and unwrapping / boxing and unboxing happens in the background. Java takes care of these details, but does not show the work.

# Open objects.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# GUI HELP

## guihelp.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)