DigiPen
INSTITUTE OF TECHNOLOGY

Chapter 1 Types and Variables

Java AP

What is a variable? A variable, just like in Algebra, is a symbol that represents a value such as:

x = 5 or y = 6

Variables can be used in place of a value in an equation: z = x + 2

we can evaluate it by substituting x with its value 5: z = 5 + 2

which results to: z = 7

In Java if we want to declare a variable and give it a value we need need to write a line of code like the following:

TypeOfVariable VariableName = VariableValue;

TypeOfVariable is the type the variable represents. This differs a bit from algebra where a variable could contain any number. In Java different types can store different types of numbers, and some cannot even store numbers at all.

VariableName (a.k.a identifier) is simply what we will refer to the value as, in the previous example this was x, y, and z. This can be anything you want in Java as long as you follow some basic rules:

- Identifiers must contain only letters (upper- or lower-case), digits (0-9), and underscores (_).
- They must begin with a letter or underscore. (Not a digit)
- There are certain words in java that have special meanings (keywords) and can't be used as identifiers. (e.g. int, double, boolean)
- Java is a case sensitive language; upper- and lower-case letters are considered to be different. (e.g. SUM, Sum, sum are all different identifiers)
- Use meaningful names, you will be glad you did.

Note: By convention identifiers for variables are lowercase. Uppercase letters are used to separate these into multiple words, for example sumOfDigits, preTaxTotal. and so on.

Some examples:

Valid	Invalid	Invalid Reason
foo1	1foo	Doesn't start with a letter or underscore
_foo	$foo	1. Doesn't start with a letter or underscore 2. $ is illegal character
foo	foo$	$ is an illegal character
valid_identifier	invalid-identifier	- is an illegal character
a_long_and_valid_name	foo bar	Can't have spaces in identifier names
Int	int	int is a keyword

Good Identifier Names	Bad Identifier Names

int rate = 60;
int time = 20;
int distance = rate * time;
	int x = 60;
	int y = 20;
	int z = x * y;

double base = 2.75, height = 4.8; double area_of_triangle = 0.5 * base * height;	double table = 2.75, chair = 4.8; double couch = 0.5 * table * chair;

Keywords

Keywords are identifiers that are reserved for the compiler. You can't use any of these as identifiers:

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

Integral Types

An integral type allows us to represent an integer. An integer is number that does not have a fractional or decimal part. For example 2, 7, 18 are integers, however 1/3 and 1.5 are not integral values. This includes negative numbers such as -1 as well. In java there are 5 types that can store integers: int, char, long, short, and byte. Each of these 5 types differ in size, this simply means the range of values they can store differs.

- **int :** This type can be used to store integral values, i.e a value with no decimal or fractional part.

```
public static void main(String[] args)
{
        int foo = 10;
        System.out.println(foo);
}
```

- **long** : stores large integral values that do not fit into an int

```
public static void main(String[] args)
{
   long l = 1000;
}
```

```
public static void main(String[] args)
{
        long l = 21 474 836 483L;
}
```

Note that a long can be designated with an L suffix this is optional if the values are between -2,147,483,648 and 2,147,483,647 if you go over those you need the L.

- **short :** fits small integral values

```
public static void main(String[] args)
{
   short s = 2500;
}
```

- **byte :** represents a single byte of data -128 to 127

```
public static void main(String[] args)
{
   byte b = 25;
}
```

- **char :** This type is usually used to store a single character

```
public static void main(String[] args)
{
   char theLetterA = 'A';
   char aValue = 0;
}
```

Here we can see that we can use two single quotes(') to make a character literal, this will be converted to an integral value for us, if we do System.out.println on a char type it will output the Unicode or ASCII(0-255) value of the number it is storing.

```
public static void main(String[] args)
{
   char theLetterA = 'A';
   System.out.println(theLetterA);
    char theLetterB = 66;
    System.out.println(theLetterB);
}
```

Output:	**A** **B**

Floating Point Types

A floating point type can store any number, positive or negative. This includes numbers with a decimal portion, like 1.5, 2.47, 3.14, along with numbers like 2 and 3. There are 2 types in Java that can represent a floating point number float and double. Each of these has slightly different precession. The less precise of the two is float.

- **float**

```
public static void main(String[] args)
{
        float pi = 3.14f;
}
```

Note that when declaring a float we add the suffix f to the name. This is required to create a float since without this suffix we end up creating a double which cannot be stored

- **double**

```
public static void main(String[] args)
{
        double pi = 3.14d;
}
```

```
public static void main(String[] args)
{
        double pi = 3.14;
}
```

- The first declaration use the ***optional*** suffix **d**. Any number with a decimal value and without a suffix is assumed to be a double, as shown in the second declaration.

Boolean Types

If you want to represent a true/false value you can use the boolean type. This is useful for writing logic such as:

if the dog is dirty
  then wash the dog

```
boolean dogIsDirty = false;
boolean dogIsNotDirty = true;
```

Initializing Variables

In all of our examples, we have declared and initialized the variable on the same line. Though, you can separate the two.

Example:

int a; /* A is only declared. Contains no value and cannot be read from */

a = 0; /* A is now initialized and can be read from */

Note: You cannot read from a variable until it has been initialized.

int a = 0; /* A is declared and initialized, can be read from and written to */

Underscores'

Note: Do not use this feature on the AP exam. Underscores are from Java 7.0 the AP exam tests 5.0, as such these are good to know for you in your own program but not for the AP exam.

When writing a number by hand we can use commas to separate the value like the following: 1,245,252. However this will not compile in Java instead we can use _ when writing out a number so 1,245,252 becomes 1_245_252

```
public static void main(String[] args)
{
        int longNum = 1_245_245_123;
        System.out.println(longNum);
}
```

Output:	**1245245123**

This is not too useful for long numbers however if you are using something like binary or hexadecimal you can separate out each byte(8 bits) or nibble(4 bits).

int nibble = 0b0101_0101;//This is only legal in Java 7 and higher

int hexByByte = 0xFF_EA;

Note: The 0b prefix tells java to interpret the values as binary, and the 0x prefix instructs java to read the value as hexadecimal. The 0b prefix is only valid in Java 7 and up and as such you should avoid it in any code you write on the AP exam. It is still good to know.

Converting Types

Certain types can be implicitly stored into other types. This occurs when the storage type is the same size or bigger than the value being stored in.

```
public static void main(String[] args)
{
        byte b = 10;
        int i = b;
        long l = i;
        float f = l;
        double d = f;
}
```

Type	Can Store
int	char, byte, short
char	char
byte	byte
short	short, byte
long	int, char, byte, short
float	int, char, byte, short, long, float
double	float, int, char, byte, short, long, float
boolean	boolean

Note: char cannot store a byte or short even though it is larger or the same size as it only stores positive numbers.

Values

Each type of variable has a range of values that it can store.

Type	Minimum Value	Maximum Value	Size (bytes)
int	-2,147,483,648	2,147,483,648	4 bytes
char	0	65,553	2 bytes
byte	-128	128	1 byte
short	-32,768	32,767	2 bytes
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	8 bytes
float	$-(2-2^{-23})\cdot 2^{127}$	$(2-2^{-23})\cdot 2^{127}$.	4 bytes
double	$-(2-2^{-52})\cdot 2^{1023}$	$(2-2^{-52})\cdot 2^{1023}$	8 bytes

A boolean is special as it holds only two values true and false. This can be represented with a single bit of data, though the size is not precisely defined.