Data Types & Variables

AP Computer Science

Credit: Slides are modified with permission from Barry Wittman at Elizabethtown College

Built-in Types of Data

Built-in types

- Today we are going to focus on four basic types
- These are:

`int`	For whole numbers
`double`	For rational numbers
`char`	For single characters
`String`	For words

- **`String`** is a little different from the rest, but we will talk about this later

The `int` Type

The `int` type

- The **`int`** type is used to store integers (positive and negative whole numbers and zero)
- Examples:
  - 54
  - -893992
  - 0

Overflow and underflow

- What happens when you add 100 to the maximum `int` value 2147483647?
- You do **not** get 2147483747
- Instead, it becomes a very negative number: -2147483549
- This phenomenon is called **overflow**
- The opposite thing happens if you have a very negative number and you subtract a number that makes it too negative
- This phenomenon is called **underflow**

Variables

- Think of a variable as a “box” you can put values into
- The name of a variable is an **identifier**
- We can **declare** a variable of type **`int`** with **identifier** **`i`** using the following line of code:

```
int i;
```

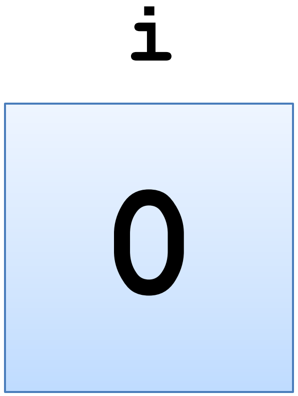
Variable Naming Conventions

- For variables, the first character is alphabetic and lowercase
- The first character of each following word should be capitalized
- An identifier must not already be in use in this part of the program
- The same rules for classes apply to variables
- **It should be meaningful!**

Assignment into an `int`

```
int i;
```

- By default, the declaration of an **`int`** puts the literal value **`0`** inside the box



- Remember, you must **declare** a variable before using it

Changing the value of a variable

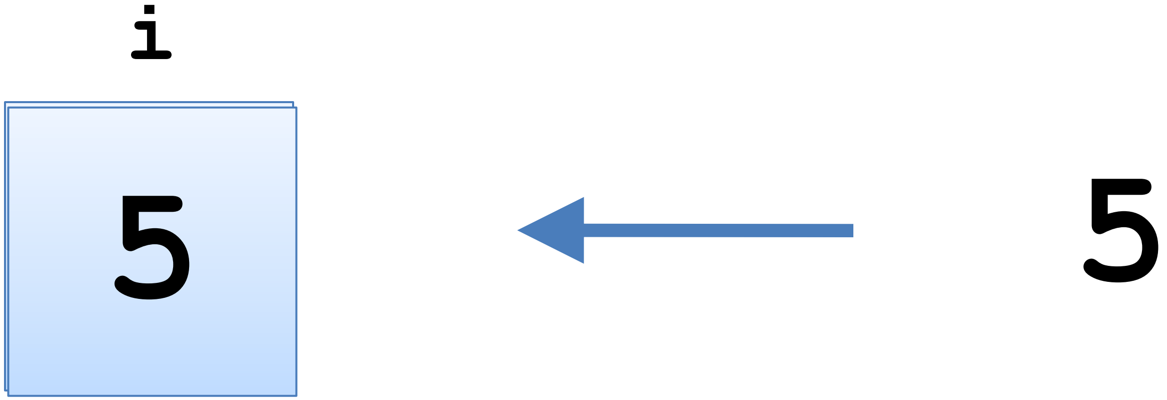
- Java variables are not like variables in math which have a fixed (but unknown) value
- Instead, a Java variable can be changed by a line of code
- We use the **assignment operator (=)** to change the value of a variable as follows:

```
int i;
i = 5;
```

Changing the value of a variable

```
i = 5;
```

- This line of code **stores 5** into **`i`**
- Think of the **`=`** operator as an arrow pointing left



- Let’s see this happen

Declaration vs Assignment

- Note the differences between declaring, assigning, and declaring and assigning
- Declaring - creates new variable with default value

```
int x;
```

- Assigning - changes value of existing variable

```
x = 10;
```

- Declaring and Assigning - creates new variable and assigns value

```
int x = 10;
```

The `double` Type

The `double` type

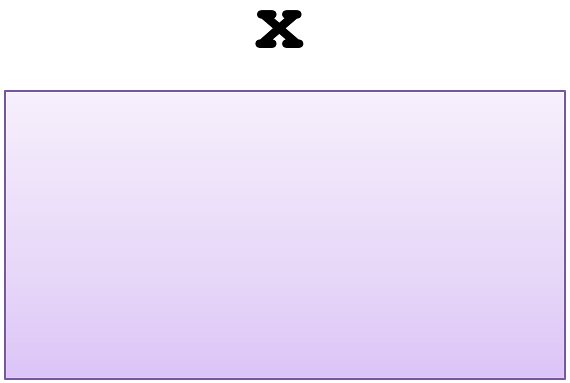
- The `double` type allows you to represent numbers with a fractional part
- Declaration of a `double` variable is like an `int` variable:

```
double x;
```

Storage for a `double`

```
double x;
```

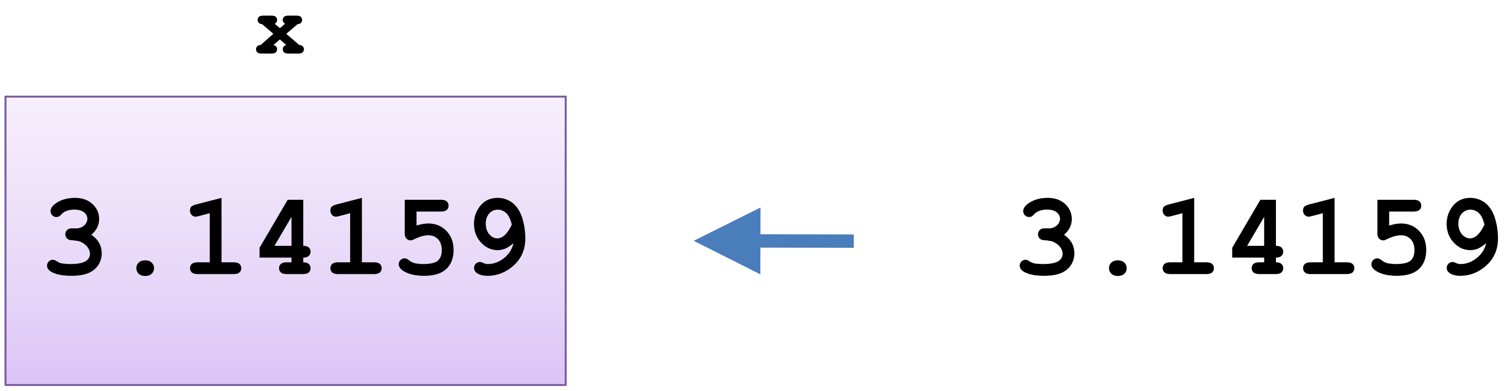
- This line of code creates a box named **`x`** designed only to hold **`double`s**



Assignment for a `double`

```
x = 3.14159;
```

- This line of code **stores** **`3.14159`** into **`x`**
- Remember that the **`=`** operator is like an arrow pointing left



`x`

`3.14159`

`3.14159`

The `char` Type

The `char` type

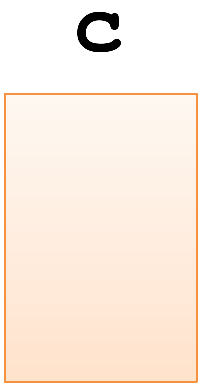
- Sometimes you need to store a single character
- This is what the **`char`** type is for
- The **`char`** type only allows you to store a single character like `'$'` or `'q'`
- You declare a **`char`** like:

```
char c;
```

Storage for a `char`

```
char c;
```

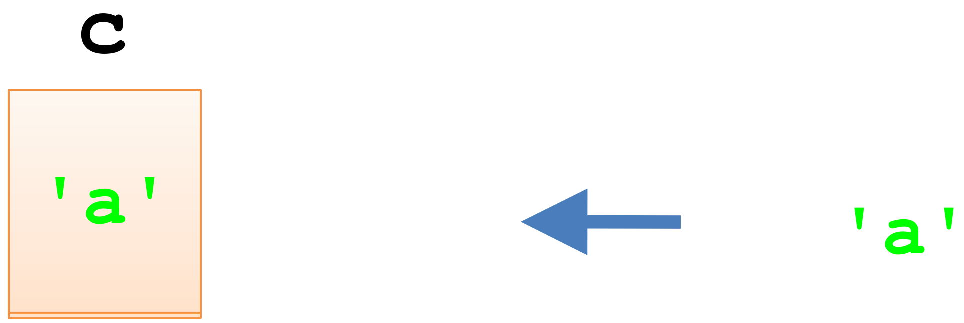
- This line of code creates a box named **`c`** designed only to hold **`char`**s
- It is used to store characters from *most* of the different scripts in the world



Assignment for a `char`

```
c = 'a';
```

- This line of code **stores** the letter `'a'` into into a variable named `c`
- We must use the single quotes so Java knows we are talking about the character `'a'` and not a variable named `a`



ASCII Characters

- ASCII is a standard used for encoding characters
- You should be able to calculate ASCII values
  - '0' - 48
  - 'A' - 65
  - 'a' - 97
- Knowing these 3 will allow you to figure out any other ASCII character

You can find the entire list of ASCII Characters here

ASCII Characters

- You can do calculations on characters

```
char one = 'a' + 1;
System.out.println(one);
```

Output
b

```
char two = 'A' + 5;
System.out.println(two);
```

Output
F

The `String` Type

The `String` type

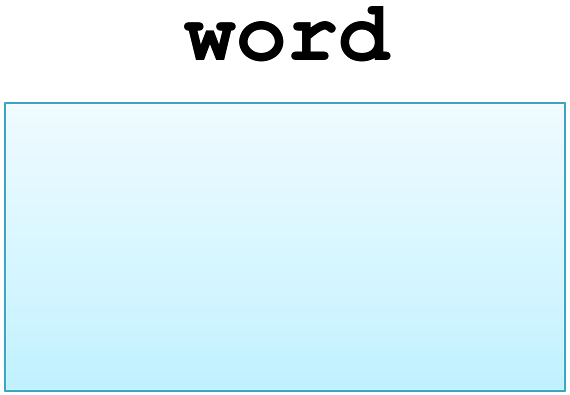
- The `String` type is different from the other types in several ways
- The important thing for you to focus on now is it can hold a large number of `char`s, not a single value
- A `String` literal is what we used in the Hello World program

```
String word;
```

Storage for a `String`

```
String word;
```

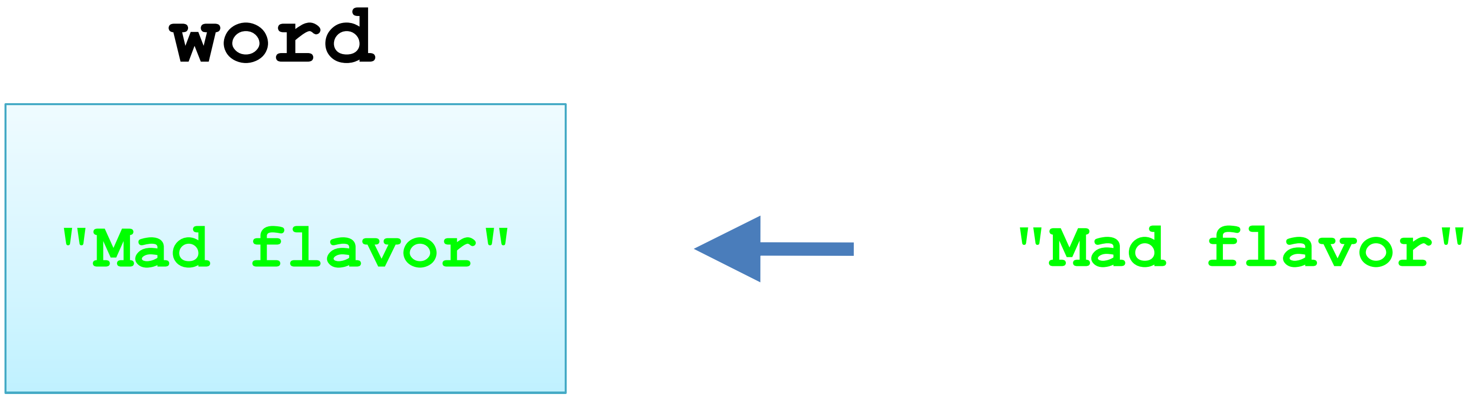
- This line of code creates a box named **`word`** designed only to hold **`String`**s
- It is used to store text of any length from *most* of the different scripts in the world



Assignment for a `String`

```
word = "Mad flavor";
```

- This line of code **stores** the **`String`** **`"Mad flavor"`** into **`word`**
- We must use the double quotes so Java knows we are talking about the text **`"Mad flavor"`**



word

"Mad flavor"

"Mad flavor"

Summary of types

Type	Kind of values	Sample Literals

`int`	Integers	`-5`
		`0`
		`900031`

`double`	Floating-point Numbers	`3.14`
		`-0.6`
		`6.02e23`

`char`	Single characters	`'A'`
		`'Z'`
		`'&'`

`String`	Sequences of characters	`"If you dis Dr. Dre"`
		`"10 Sequipedalians"`

constants

Constants

- Often in a program you want to give a name to a constant value.
- For example you might have a tax rate of 0.045 for durable goods and a tax rate of 0.038 for non-durable goods.
- These are constants, because their value is not going to change during a run of the program.

```
final static double DURABLE = 0.045;
final static double NONDURABLE = 0.038;
```

- The reserved word final tells the compiler the value will not change.

Operations on `ints`

The + Operator for `int`

- Use the **+** operator to add two `int`s together

```
int a;
int b;
a = 5 + 6;   // a contains 11
b = a + 3;   // b contains 14

a + b;       // not allowed, does nothing

a = a + 1;   // a contains 12, and b?
```

Shortcuts

- Some expressions are used so often, Java gives us a short cut
- `x = x + y;` can be written `x += y;`
- `x = x + 1;` can be written `x++;`

```
int x;

x = 6;            // x contains 6
x += 4;           // x contains 10

x++;              // x contains 11
```

The – Operator for `int`

- Exactly like + except performs subtraction

```
int a;
int b;
a = 5 - 6;   // a contains -1
b = 3 - a;   // b contains 4

a -= 10;     // shortcut for a = a – 10;

a--;         // shortcut for a = a – 1;
```

The * Operator for int

- The * operator performs multiplication

```
int a;
int b;
a = 5 * 6;   // a contains 30
b = a * 3;   // b contains 90

a *= 2;      // shortcut for a = a * 2;
```

The / Operator for `int`

- The / operator performs **integer** division
- **Not** the same as regular division

```
int a;
int b;
a = 3;       // a contains 3
b = a / 2;   // b contains 1

a /= 2;      // shortcut for a = a / 2;
```

- The fractional part is dropped, **not** rounded

The % Operator for int

- The % operator is the mod operator
- It finds the remainder after division

```
int a;
int b;
a = 8;       // a contains 8
b = a % 5;   // b contains 3

a %= 2;      // shortcut for a = a % 2;
```

- This operator is a good way to find out if a number is even or odd

Operations on `doubles`

The + Operator for `double`

- Exactly the same as **+** for **`int`**, except now you can have fractional parts

```
double a;
double b;
a = 3.14159; // a contains 3.14159
b = a + 2.1; // b contains 5.24159

a += 1.6;    // shortcut for a = a + 1.6;

a++;         // shortcut for a = a + 1.0;
```

The – and * Operator for double

- No surprises here
- They do subtraction and multiplication

```
double a;
double b;
a = 3.14159;        // a contains 3.14159
b = a - 2.1;        // b contains 1.04159

a = b * 0.5;        // a contains 0.520795
```

The / Operator for `double`

- Unlike **`int`**, this division does have fractional parts

```
double a;
double b;
a = 3;            // a contains 3.0
b = a / 2;        // b contains 1.5

b = 3 / 2;        // b contains 1.0
```

- Can you explain this mystery?

Complex expressions

- How complex can expressions get?

```
int a = 31;
int b = 16;
int c = 1;
int d = 2;

a = b + c * d - a / b / d;
```

- What is the value of a?
- 18!

Complex expressions

- Order of operations holds like in math

```
int a = 31;
int b = 16;
int c = 1;
int d = 2;

a = (((b + c) * d) - a / b) / d;
```

- You can use parentheses to clarify or change the precedence
- Now **`a`** is 16

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
multiplicative	* / %
additive	+ -
assignment	= += -= *= /= %=

This is a sample of the entire list of operator precedence. You can find the entire list located **HERE**.

Casting

- You cannot directly store a **`double`** value into an **`int`** variable

```
int a = 2.6; // fails!
```

- However, you can cast the **`double`** value to convert it into an **`int`**

```
int a = (int)2.6; // succeeds! (a = 2)
```

- Casting tells the compiler you want the loss of precision to happen
- You can always store an **`int`** into a **`double`**

Rounding

- In Java, the conversion of a **`double`** into an **`int`** does not use rounding
- As in the case of integer division, the value is always rounded down
- You can think of this as using the **floor** function from math
- If you want to **round** normally, you can simply add 0.5 before the cast

```
double x = 2.6;
int a = (int)(x + 0.5);    // rounds
```