Data Types in Java

- Data is the information that a program has to work with.
- Data is of different *types*. The type of a piece of data tells Java what can be done with it, and how much memory needs to be put aside for it.

- When we create a variable in Java, we need to specify:
  - the type of the value we want to put in there, and
  - the name we will use for that variable.

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared*, specifying the variable's name and the type of information that will be held in it


```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

```
int count=1, temp=0, result;
```

Variables can also be given initial values

2

Constants

- A constant is an identifier that is similar to a variable except that it holds one value for its entire existence
- The compiler will issue an error if you try to change a constant
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

- Constants:
  - give names to otherwise unclear literal values
  - facilitate changes to the code
  - prevent inadvertent errors

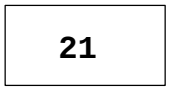
Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

Type	Storage	Min Value	Max Value
byte	**8 bits**	**-128**	**127**
short	**16 bits**	**-32,768**	**32,767**
int	**32 bits**	**-2,147,483,648**	**2,147,483,647**
long	**64 bits**	**< -9×10^{18}**	**> 9×10^{18}**
float	**32 bits**	**+/- 3.4×10^{38} with 7 significant digits**	
double	**64 bits**	**+/- 1.7×10^{308} with 15 significant digits**	

Type *int* *(byte, short, & long)*

- An `int` is a whole number (e.g. 21). You can do arithmetic with an `int`.
- `int age = 21;`



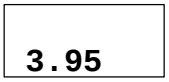
age

- addition +
- subtraction -
- multiplication *
- division /
- modulus %

```
int age = 15;
age + 3
2 * age - 4
age / 2
age % 10
```

Type `double` (float)

- The type called `double` is used to store a real number , i.e. a number with a decimal point. It is stored in a different format from an `int`.
- You can do arithmetic on a `double`.
- `double price = 3.95;`



price

addition +
subtraction -
multiplication *
division /

```
price * 0.10
price / 2
price + price * 0.1
```

Type *boolean*

- A `boolean` value represents a true or false condition
- A boolean can also be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

7

Type *char*

A `char` is a character, i.e. a bit pattern you can produce by pressing a key (or a combination of keys) on a keyboard.

Examples are

```
'a' 'A'  '3' '?' '!'
```

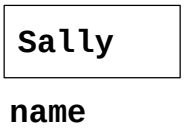
```
char response = 'Y';
```

You cannot do arithmetic on a `char`.

A `String` is a collection of `char`s.

Type *String*

- A `String` is a collection of characters (e.g. `"Sally"`).
  - `String name = "Sally";`



- A `String` value is always written in double quotes. You can have an empty `String`, shown as `""`.
- A `String` has many methods including:
  - change itself to upper/lower case
  - tell you how long it is (how many characters)
  - give you the character at a specified position
  - the string concatenation operator `+`

Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can always be used to force the evaluation order

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4

a + b * c - d / e
  3   1   4   2

a / (b + c) - d % e
  2   1   4   3

a / (b * (c + (d - e)))
  4   3   2   1
```

Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4     1   3     2
```

Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)

Data Conversions

- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - Only widening conversions can happen via assignment
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

Data Conversions

- *Casting* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (float) total / count;
```