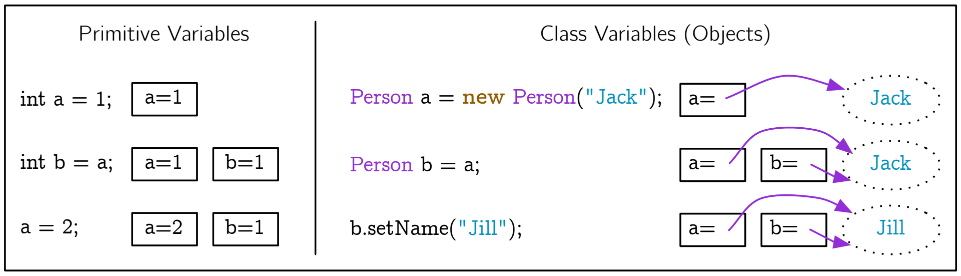
Types

The first time you use a variable in Java, you must tell the Java compiler its type (like `int`, `float`, or `String`). This is called *declaring* the variable. **A variable exists only inside the brackets it was declared in.** As long as it exists, the variable may not be re-declared and hence can never change type.

A variable's type can be either a *primitive* (like `boolean`, `int`, or `char`) or a *class* (like `String`, `ArrayList`, or `File`). Unlike classes, primitives are built into Java itself and you cannot add your own.

Warning: Primitives and classes have completely different copy behaviors! Primitives *copy-by-value*, meaning that after the copy both variables have independent existences. Classes *copy-by-reference*, which means both variables are really just pointers to the same object out in memory.



Loops

Java has three kinds of loops. Secretly though, they're all just `while` loops in disguise.

1. `while` loops will repeat as long as a condition is satisfied at the beginning of each loop.

```
while (BOOLEAN EXPRESSION)
{
    CODE TO REPEAT
}
```

   A *boolean expression* is anything that evaluates to type `boolean`. This can be

   - A test (like `x < 10`).
   - A boolean variable (like `done` (assuming `done` was defined with type `boolean`)).
   - A boolean function (like `isPrime(x)`). We'll get to those on the next page.

2. `for` loops are just shorthand for a very common type of `while` loop.

```
for (INITIALIZER; CONDITION; INCREMENT)
{
    CODE TO REPEAT
}
```

```
INITIALIZER;
while (CONDITION)
{
    CODE TO REPEAT
    INCREMENT;
}
```

3. `for .. in` loops are the most like Python loops. They are just shorthand for a special type of `for` loop.

```
for (TYPE var in
iterable)
{
    CODE TO REPEAT
}
```

```
for (Iterator<TYPE> it = iterable.iterator(); it.hasNext();)
{
    TYPE var = it.next();
    CODE TO REPEAT
}
```

   Any of the standard collections (like lists, arrays, or dictionaries) you encounter in Java will be *iterable*. If you're curious about how they work or want to make your own class iterable, Google "java iterators."

Functions

Java functions have a few more bells and whistles than their Python equivalents. And you can't avoid them. The entry point to every Java program is a very scary function called `main`. Here it is:

$$\underbrace{\texttt{public}}_{\texttt{visibility}}\ \underbrace{\texttt{static}}_{\texttt{???}}\ \underbrace{\texttt{void}}_{\texttt{return}}\ \underbrace{\texttt{main}}_{\texttt{name}}\ \underbrace{\texttt{(String[] args)}}_{\texttt{arguments}}$$

Surprise: It's not actually random gibberish! And believe it or not, `main` is about as bad as Java functions get.

- `public` means that the function can be called from outside the class it lives in. This is called *visibility*. Java will yell at you if `main` isn't `public`, but you can make other functions `private` (only accessible within the class itself) or `protected` (accessible to all subclasses as well) to protect your code from those silly users.
- `static` means the function belongs to the whole class, rather than a particular instance of the class. There is no `nonstatic` keyword; all functions are assumed to be non-static by default. See if you can guess which of these functions is `static`:

  ```
  print("Bob is named "+bob.name());
  ```

  ```
  print("Average height is "+Person.height());
  ```

- `void` is the *return type* of the function. `void` is a special type that means "this function doesn't return anything." Here's a function that actually returns something:

  ```
  public boolean isPalindrome (String word)
  ```

- `main` is the name of the function.
- `String[] args` means that the function takes a single argument of type `String[]` named "`args`." "`[]`" is Java-ese for an array, so `String[]` means "an array of strings." Java uses this string array to tell your program what arguments the user typed into Terminal. This is basically what Java is doing under the hood:

  ```
  >> java AreAnagrams elvis lives
  ```

  ```
  main(["elvis","lives"])
  ```

That's just about all there is to functions. Not quite though. Because every function knows the types of its arguments, you can have multiple different functions with the same name. This comes in handy in situations like this:

```
public static void print (String x) { Just print x. }
public static void print (int x) { Convert x to a string and then print it. }
public static void print (float x) { Convert x to a string and then print it. }
...
```

Now you call `print(x)` without worrying about what type it is! The Java compiler will pick the right one for you.