If we could run this code, Java would find the `charAt` method in `o`, since it refers to a `String` object:

```java
Object o = new String("hello");
char c = o.charAt(1);
```

But the code won’t compile because the compiler cannot be sure it will find the `charAt` method in `o`.

Remember: the compiler doesn’t run the code. It can only look at the type of `o`.

So we need to cast `o` as a `String`:

```java
char c = ((String) o).charAt(1);
```
Javadoc

Like a Python docstring, but more structured, and placed above the method.

/**
 * Replace a square wheel of diagonal diag with a round wheel of diameter diam. If either dimension is negative, use a wooden tire.
 * @param diag Size of the square wheel.
 * @param diam Size of the round wheel.
 * @throws PiException If pi is not 22/7 today.
 */

public void squareToRound(double diag, double diam) { ... }

Javadoc is written for classes, member variables, and member methods.
This is where the Java API documentation comes from!

In Eclipse: Project → Generate Javadoc
Java naming conventions

The Java Language Specification recommends these conventions

Generally: Use camelCase not pothole_case.

Class name: A noun phrase starting with a capital.

Method name: A verb phrase starting with lower case.

Instance variable: A noun phrase starting with lower case.

Local variable or parameter: ditto, but acronyms and abbreviations are more okay.

Constant: all uppercase, pothole.

E.g., MAX_ENROLMENT
Direct initialization of instance variables

You can initialize instance variables inside constructor(s).

An alternative: initialize in the same statement where they are declared.

Limitations:

Can only refer to variables that have been initialized in previous lines.

Can only use a single expression to compute the initial value.
What happens when

1. Allocate memory for the new object.

2. Initialize the instance variables to their default values:
   - 0 for ints, \texttt{false} for booleans, etc., and \texttt{null} for class types.

3. Call the appropriate constructor in the parent class.
   - The one called on the first line, if the first line is \texttt{super(arguments)}, else the no-arg constructor.

4. Execute any direct initializations in the order in which they occur.

5. Execute the rest of the constructor.
Abstract classes and interfaces

A class may define methods without giving a body. In that case:

Each of those methods must be declared `abstract`.

The class must be declared `abstract` too.

The class can’t be instantiated.

A child class may implement some or all of the inherited abstract methods.

If not all, it must be declared `abstract`.

If all, it’s not abstract and so can be instantiated.

If a class is completely abstract, we may choose instead to declare it to be an `interface`.
Interfaces

An interface is (usually) a class with no implementation. It has just the method signatures and return types. It guarantees capabilities.

Example: java.util.List

"To be a List, here are the methods you must support."

A class can be declared to implement an interface. This means it defines a body for every method.

A class can implement 0, 1 or many interfaces, but a class may extend only 0 or 1 classes.

An interface may extend another interface.
Generics: naming conventions

The Java Language Specification recommends these conventions for the names of type variables:

very short, preferably a single character

but evocative

all uppercase to distinguish them from class and interface names

Specific suggestions:
Maps: \( K, V \)
Exceptions: \( X \)
Nothing particular: \( T \) (or \( S, T, U \) or \( T_1, T_2, T_3 \) for several)