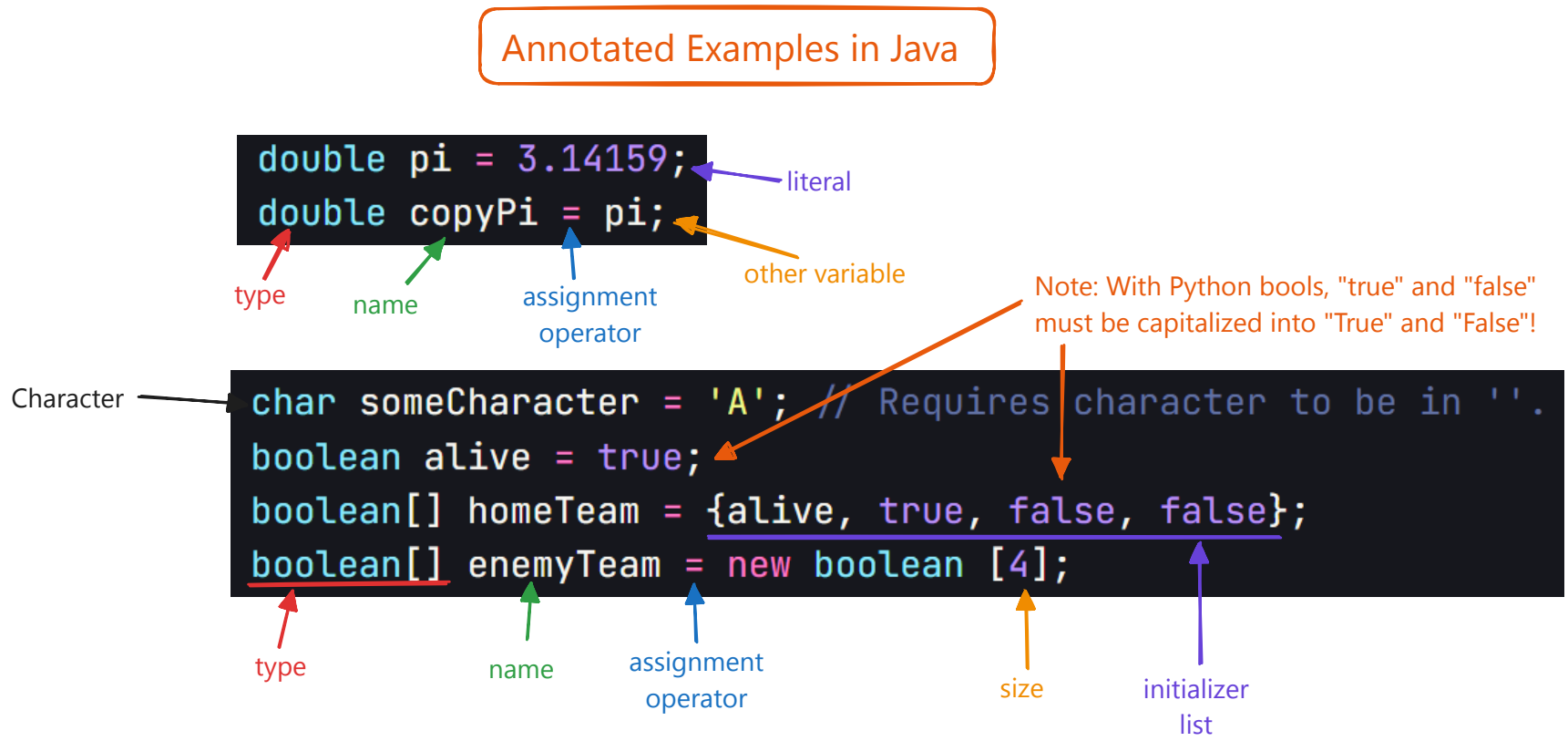
Java Cheatsheet

Variables



Scanner

```java
import java.util.Scanner; // Required to use the scanner.

// ...

Scanner scan = new Scanner(System.in); // Initialize a scanner.

String s = scan.nextLine();      // `nextLine` gets a string.
int num = scan.nextInt();        // `nextInt` gets an integer.
double num2 = scan.nextDouble(); // `nextDouble` gets a floating-point.

scan.close(); // Remember to close the scanner.

// Refer to the slide in day 2 about buffers if your strings come back empty.
```

Operators

Operators	Precedence
postfix	*expr++* *expr--*
unary	*++expr* *--expr* *+expr* *-expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= *instanceof*
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Casting

```java
double num = 3.14159;
int num2 = (int)num; // `num` casted to an integer.

// Casting to a String is a bit different:
String numString = Double.toString(num); // `Double` is uppercase!
String numString2 = Integer.toString(num2);

// Casting from a String to other data types:
String numString3 = "123";
int num3 = Integer.valueOf(numString3);
```

Arrays

```java
int[] arr = {5, 6, 7};
int i = arr[0]; // First element: 5
int j = arr[1]; // Second element: 6
// ...

int size = arr.length; // Number of elements in array.

int[][] matrix = { // Multidimensional
    {2, 4, 6},
    {8, 10, 12},
    {14, 16, 18}
};
```

Strings

```java
String s = "Hello, world!";
char c = s.charAt(7); // Get character at index 7.

// Get a substring from string `s`, starting at index 0 and stopping right before index 5.
String sub = s.substring(0, 5);

String s1 = "Hello, ";
String s2 = "world!";
String s3 = s1 + s2; // ⟶ s3 is "Hello, world!"
```

Basic Escape Sequences

`\"`, `\'`, `\\`, `\n`, `\t`, ...

Functions

Global Variable

`final` in this context indicates a mathematical constant

```
// Global variable used by `volumeOfCylinder`.
private static final double pi = 3.14159;
```

Function with return value and parameters

Return Type, Function Name, Parameters Types

```
// Function with return value.
private static double volumeOfCylinder(double radius, double height) {
    double baseArea = pi * radius * radius;
    double volume = baseArea * height;
    return volume;
}
```

Parameter Variable Names

Return Value

Function Body

Function with no return value or parameters

Return type `void` indicates no return value

Function Name

```
// Function without return value.
private static void cow() {
    // Note: we are using escape sequences here so that we can use backslash (\) characters inside strings.
    System.out.println("^__^             ");
    System.out.println("(oo)\\_______     ");
    System.out.println("(__)\\       )\\/\\");
    System.out.println("    ||----w |    ");
    System.out.println("    ||     ||    ");
}
```

No parameters --> empty parenthesis ()

Function Body

No return value

Calling Functions

Arguments --> Parameters

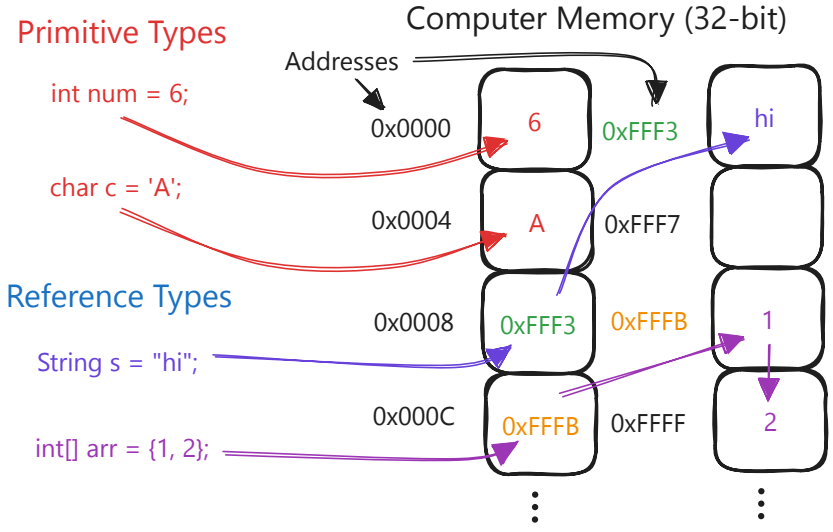
Function Call

```
double r = 27;
double volume = volumeOfCylinder(r, 10);
cow();
```

Calling a void function

Return value is assigned to new variable

By Reference



Comparing Primitive Types

```
int num1 = 5;
int num2 = 6;
```

num1 → 5
num2 → 6

num1 == num2

5 == 6

false

Comparing Reference Types

Incorrect

```
String s1 = "hello";
String s2 = "hello";
```

(they are the same)

s1 → 0x1234 → hello
s2 → 0xBEEF → hello

s1 == s2

0x1234 == 0xBEEF

false

(we get false even if the data of the strings is the same)

Correct

```
String s1 = "hello";
String s2 = "hello";
```

(they are the same)

s1 → 0x1234 → hello
s2 → 0xBEEF → hello

s1.equals(s2)

.equals method compares "hello" to "hello"

true

Comparison Operators

- == → equality operator, tests if two values are equal
- The single equals sign = is already used as the assignment operator!
- != → inequality operator, tests if two values are not equal
- < → less than
- > → greater than
- <= → less than or equal to
- >= → greater than or equal to

Logical Operators

- `&&` → AND operator
- `||` → OR operator
- `!` → NOT operator

Conditionals

```
if (/* boolean expression */) {
    // ...
} else if (/* boolean expression */) {
    // ...
} else {
    // ...
}
```

Loops

```
int num = 0;
while (num < 10) {
    System.out.print(num + " ");
    ++num; // This is the prefix increment operator.
    // It does the same thing as num = num + 1.
}
```

```
//   init.      cond.    update
for (int i = 0; i ⩽ 20; i += 2) {
    System.out.print(i + " ");
}
```

```
String[] arr = {"Alice", "Bob", "Charlie"};
for (String s : arr) {
    System.out.print(s + " ");
}
```

```
String input;
do {
    System.out.print("Please enter \"confirm\": ");
    input = scan.nextLine();
} while (!input.equals("confirm"));
```

Loop statements: `break`, `continue`

Classes

```java
public class ClassExample {
    public static void main(String[] args) {
        // We can still use arrays to store multiple people, but here we only need one array.
        Person[] people = {
            // Use the `new` keyword when creating objects from the Person class.
            new Person("Alice", 20, "Shanghai", "Aly"), // This a person object.
            new Person("Bob", 21, "New York", "B"), // Another person object.
            new Person("Charlie", 25, "London", "Chip") //  ...
        };

        // Get a person.
        Person person = people[0]; // `person` is an object instantiated from `Person`.
        Person samePerson = people[0];

        person.sayHi();
        samePerson.sayHi();

        // Both `person` and `samePerson` refer to the same object of `Person`.
        // Changing one variabe's attributes will also be reflected by the other variable!
        person.city = "Wonderland";
        samePerson.sayHi();
        // This is why they classes are "reference types".

        // We can access static attributes and static methods directly from a class without needing an object.
        System.out.println(Person.NUMBER); // Access `NUMBER` directly from `Person`.
        Person.someStaticMethod(); // Call `someStaticMethod` directly from `Person`.

        // You can treat the keyword `null` as meaning "no data specified yet".
        Person temporaryPerson = null;
        // The object can be properly initialized later.
        // temporaryPerson = new Person("Echo", 1000, "Atlantis", "E");
        // Calling methods on an object with value `null` will cause a runtime error!
        // temporaryPerson.sayHi();

        // Attempting to access private members will result in an error!
        // String fail = person.nickname;
        // person.sayNickname();
    }
}

// The Person class.
class Person {
    // A static attribute.
    public static final int NUMBER = 10;

    // Attributes:
    public String name;
    public int age;
    public String city;

    // A private attribute.
    private String nickname;

    // A constructor. Special method that does not require specifying a return type.
    public Person(String name, int age, String city, String nick) {
        // We use the `this` keyboard to distinguish between the `name` argument and the `name` attribute.
        this.name = name;
        this.age = age;
        this.city = city;
        // `this` is not required for the `nickname` attribute because the argument is called `nick`.
        nickname = nick;
    }

    // A method, just like a function.
    public void sayHi() {
```

Other Data Structures

```java
import java.util.Arrays;
import java.util.ArrayList;

// ...

ArrayList<Integer> nums = new ArrayList<>(Arrays.asList(2, 4, 6, 8));

// Get / set an element at a specific index.
int element = nums.get(2); // Gets the integer at index 2 ⟶ 6
nums.set(1, 9); // Sets the element at index 1 to the value 9.

// Add an element to the end of the array-list.
nums.add(5);

// Insert an element at index 3.
nums.add(3, 6);

// Remove an element at index 1.
nums.remove(1);

// Get the number of elements in the linked-list.
int count = nums.size();
```

```java
import java.util.Arrays;
import java.util.LinkedList;

// ...

LinkedList<Integer> nums = new LinkedList<>(Arrays.asList(2, 4, 6, 8));

// Get / set an element at a specific index.
int element = nums.get(2); // Gets the integer at index 2 ⟶ 6
nums.set(1, 9); // Sets the element at index 1 to the value 9.

// Other methods.
element = nums.getFirst();
System.out.println(element);
element = nums.getLast();
System.out.println(element);

// Add an element to the beginning and end of the linked-list.
nums.addFirst(5);
nums.addLast(5);

// Insert an element at index 3.
nums.add(3, 6);

// Remove an element at index 1.
nums.remove(1);

// Remove the first and last elements.
nums.removeFirst();
nums.removeLast();

// Get the number of elements in the linked-list.
int count = nums.size();
```

```java
import java.util.HashMap;

// ...

// Key type: `String`
// Value type: `Integer` / `int`
HashMap<String, Integer> map = new HashMap<>();

// Add / change a key-value pair.
map.put("Alice", 2000);
map.put("Bob", 1900);
map.put("Charlie", 1950);

map.put("Alice", 3000); // Changes the Alice's existing balance.

// Check if the map has a specific key.
System.out.println("Contains Alice: " + map.containsKey("Alice"));
System.out.println("Contains Dave: " + map.containsKey("Dave"));

// Get a value associated with a key.
System.out.println("Alice's Balance: " + map.get("Alice"));
System.out.println("Dave's Balance: " + map.get("Dave")); // `null` since Dave is not in the map.

// Get the number of key-value pairs in the hashmap.
int count = map.size();
```

Exception Handling

```java
// Start of the `try` block!
try {

    // ArrayIndexOutOfBoundsException
    int[] nums = {1, 2, 3};
    System.out.println(nums[10]); // Index 10 is out-of-bounds.

    // ArithmeticException
    int n = 10 / 0; // Dividing by 0 is illegal.

    // NullPointerException
    String s = null;
    s.length(); // Don't use an object which is `null`.

    // Your own exception!
    throw new Exception("MY EXCEPTION");

// End of the `try` block. Start of the `catch` block!
} catch (Exception e) {
    System.out.println("A problem occurred!");

    // You can also print the details of the exception.
    System.out.println(e.toString());
    System.out.println(e.getMessage());
}
```

Packages

Refer to slides 140 - 144 on day 5 for how to use packages.