Domain 2: Data Types, Variables, and Expression Videos

variable → container for storing data
primative → pre defined

Variable Decleration:

type name = value; → init. variable
or type name;

Naming: → -camel
- letters/num/currency
- cannot start with #

Type

byte → -128 → 127
char → single character or positive number (0 → 2^{16})
- if letter is used must be in single quotes ''
- stands for character

int → integer (32 bit whole number)
- whole number
- (-2B - 2B)

double → double (64 bit - decimal number)
- ($-2^{1074} \rightarrow 2^{1023}$)

short → 16 bit whole number
- ($-2^{15} \rightarrow 2^{15}-1$)

long → 64 bit whole number
- needs "l" after the whole number to signify (L)ong otherwise considered int
- ($-2^{63} \rightarrow 2^{63}-1$)

float → 32 bit decimal
- need "f" after numba or java reads as double

boolean → t/f values

Wrapper Types

- collections only store objects not primitives
  ex. ArrayList<int> not poss. while ArrayList<Integer> is
- wrapper types can be null
- wrappers have useful utility methods

Strings

- declaration

  String name = "Shivansh";

- Manipulation
  - str 1 + str 2 ← Comparison
  - str 1 + str 2 ← concatination
  - str. to Upper Case() } switches cases
    str. to Lower Case() }
  - str. length() ← returns length of string
  - str. is Empty () ← checks if empty
- Formatting
  - String. format ("|%os|", product 2)
    ↳ %os means display as string
  - Control length example:
    - String. format ("|%20s|", product 2)
    - output

      str

      takes 20 char
    - useful example is if you have a list that you want to output and line up.
- Operators
  - str. trim();
    ⇒ cuts out extra spaces at the end of string
  - str. substring (start, end);
    ⇒ cuts out a specific substring starting at character specified by start int. then ending with character specified by end int.
  - str. charAt (Index);
    ⇒ gives the character within the string specified by the index

Strings (Continued)

- Conversion
  - Method I
    - => Wrapper.toString();
    - => Can be used with primitive type classes, converting them to string
    - => Can be a nuisance because wrapper class is mandatory
  - Method II
    - => String.valueOf(var);
    - => works with all primative data types
    - => Much easier than "wrapper.toString()" because no need to find wrapper class
- ALL strings are immutable
  - => Immutable means that once the object is declared, it cannot be changed. In order to change it a new object must be created with the same name. Behind the scene, the original variable is deleted
  - -> Why?
    - thread safety without sychronization
    - easy to reason about
    - fewer bugs due to accidental mutation
    - used in functional programming and streams

Null

=> means $\emptyset$ or empty

-> * When console printing variable with null, Java outputs "null" *

-> ex.

```
String dummy;
System.out.print(dummy);
```

} still will print an error, need to set = null, ex below

-> ex.

```
String dummy = null;
System.out.print(dummy);
```

} output is null

Arrays

=> containers with multiple compartments to store information

- Decleration => denoated by [] after var. type

· One-Dimmensional

- ex. `String [] students = new String [4];`

```
students [0] = "a";
students [1] = "b";
students [2] = "c";
students [3] = "d";
```

- 0 based
- single element access ex.

- looping ex.

```
for ( String student: students){
    System.out.println (student);
}
```

· Two Dimmensional

- ex. `String [] [] names = new String [n1] [n2];`

```
names [0][0] = "a1";
names [0][1] = "a2";
names [0][2] = "a3";
names [1][0] = "b1";
names [1][1] = "b2";
names [1][2] = "b3";
```

ex. array:

a1	a2	a3
b1	b2	b3

· Fast init of Onedarray

- ex. `String [] names = {"a1", "a2", null, "a3"};`
- differences:
  - {} instead of []
  - no need for decleration of new string
  - no need for size decleration
  - size is still fixed

Arrays (continued)

- Dynamic Array
  - declaration → add datatypes for <>
    - ex. `ArrayList<String> names = new ArrayList<>();`
      `names.add(v1);` } Adds v_1 v_2 to names
      `names.add(v2);`
      `names.get(i);` → i being the index
  - iteration
    - `Iterator names = names.iterator();`
      `while (names.hasNext()) {`
      `System.out.println(names.next());`
      `}`
    - must use Iterator object
  - methods:
    - arr.add(v) → adds value to end
    - arr.get(i) → gets value at index
    - arr.remove(v) → removes VALUE
    - arr.remove(i) → removes INDEX

Data Conversion

- implicit conversion to looser data type
- ex.

```
double a = 12.9;
int    b = 4;
int    c = (int)(bagSize);   → Only works with primative dts.
String d = String.valueOf(a); → Object data types
```

`sop(c);` //12 → chops off decimal
`sop(d);` //12.9 → direct conv. to string
`sop(a*b);` //51.6 → converts to double

Data Conversion (continued)

ex.

```
String str = new String("4.8");
Double d = Double.parseDouble(str); -> works
int i = Int.parse Int(str); -> error bc. 4.8 ≠ int
```

Arithmetic Operators

+	add	+=	add	assigns val
-	sub	-=	sub	
*	mult	*=	mult	
/	div	/=	div	
%	mod -> gives rem.			

Parentheses
Exponents
Multiplication
Division
Addition
Subtraction