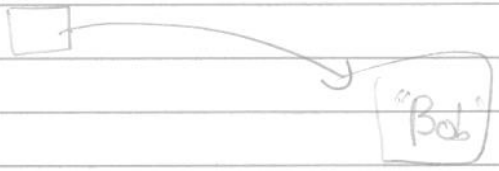
Ch. 6: Arrays & Arraylists

9/28/17

- Data structures - collections of related data items
- array objects - data structures consisting of related data items of the same type
- Java's types are divided into primitive types & reference types.
- All non-primitive type are reference types.
  - Start w/ capital letter. ex: String, Scanner
- Prog. use var. of ref. types (normally called references) to store the address of objects in Comp.'s memory.

ex: `String foo = "Bob";`



- array - group of variables (called elements / components) containing values of same type.
- Arrays are objects, they're considered ref. types
- Elements either prim. or ref. types.
- 1st element in an array is @ Zero. (zeroth slot)
- a prog. refers to an array's element w/ an array-access
- name of array followed by index of particular element in brackets.

Ex:

int ar: `int [] c = new int [10]`

`c [0] = 20;`

String ar: `String [] d = new String [23];`

public - access outside of class in which it's defined
private - " w/in defined class
void - returns nothing

Name of parameter
↓
`(String [] args)`
↳ defines a type

cp file 1 file 2
↓ ↓
prog name Command-line arguements

- Every array object knows how big it is, & is stored in a length instance variable

- to create an array object, you may specify the type the array elements + # of elements as part of an array-creation expression.

- You can create an array & initialize its element w/an initializer. - separated by commas.

- final applied to variable, can't be changed once set.
  ↳ in ALL caps

pg. 201

10/6/17

- prog use counter variables to summarize data, such as results of a survey.

ex: 20 students 1 - 5
array out of bounds - 14

exception - something generated in program that's thrown & caught, handled to not crash program

try/catch block:
↳ single/block statements - could cause exception. If error in try, it will look for match in Catch.

- more than 1 catch block

- enhanced for-statement

- for (parameter : ArrayName
↳ type & identifier

pass array into method, array needs declared variable

- reference - like a memory address

- pass-by-value: copy of value is pass to called method

" " - reference: called method can access args. value directly & modify data.

- every dimention, there's index

- two-dimentional arrays : more that 2 dimentions
- nested array initializers
  - int [][] b = {{1, 2}, {3, 4}};

ex: int [][] a = new int [5][2]

- variable-length argument list : create methods that receive unspecified # of arguments

ex: public static void foo (int... a) → 2
    int [] a;

- ellipsis (...)

10/9/17

```
public class ...
}

    main method
    }

        " velocity = Double.parseDouble
    double [] pCoords = new double [2];
        "  " pVelocity = new  "  [2];

        "     initialHeight = 4.0;

    pCoords [0] = 0.0;
        "   [1] = initalHeight;

    double  velocity = 10.0;
        "     angle = 30,
    calculateProjectileVelocity(velocity, angle, pVelocity);
    System.out.println("x part of velocity: " + pVelocity [0]
    {   "  "  "  " ("," "  "  "  :  "  +  "  [1];
```

→

```
    public static void calculateProjectileVelocity (double cp, double
angle, double [] velocity)
    {

        double angleInRads = angle * Math.PI /180.0;

        "     x = cp * Math.cos(angleInRads);
        "   y = "  *  "  .sin( "  "  " );

    velocity [0] = x;
        "    [1] = y;
```

When running 'java Projectile [ ] more

* pipe = take 1 thing & put it to another

after while:

updateProjectileVelocity (timeStep, pVelocity)

Public Static Void ↓ (double dt,

time += timeStep;

double timeStep = 0.1;

time = 0.0;

pCoords[2]

While (time < 0.0) {

updateProjectileCoordinates (timeStep, pVelocity, pCords)

Sys: out. println ("x: " + pCoords[0] " y: " + pCoords[1] + time

Public static Void ↓ (double dt, double [] vel, double [] coords

coords [0] = coords [0] + dt * vel [0]; > for (int i=0; i<2, i++)

coords [1] = coords [1] + dt * vel [1]; coords[i] += dt * vel[i]

10/11/17

String [ ] → array of strings

for (int i = 0; i < numberOfTimes; i++)

System.exit → quits program

* Objects:

integer & integer → date

instance variables - generalization

constructor

① What it knows
② " " can do