AP Computer Science – Cram Sheet

- **`System.out.println`** – don't abbreviate as SOP
- **No user input** – never write `in.nextLine()` , `in.nextInt()` , etc
- **Don't make up variable names!** – parameters and instance variables should be used. Don't make up variable names for information that should be provided to you. Only make up variable names for things that are counters or calculations or temporary variables.
- **Order of Operations** – `()` then `*/%` then `+–`, evaluate left to right
- **PLOP (Possible Loss of Precision)** - a `double` cannot be stored in `int`
- **Integer Division** – truncates answer (drops decimal)
  - **Double Division** – doubles "take over"; same for *, +, -, etc with doubles
    `int/double = double` `double/int = double`
- **Concatenation** – numbers joined to strings are just tacked onto end
  - String + number + number = tack on strings
  - `"Hello" + 2 + 3 = "Hello23"`
  - `"Hello" + (2 + 3) = "Hello5"`
  - `2 + 3 + "Hello" = "5Hello"`
- **`== vs .equals`** – `.equals()` for `Strings/Objects, ==` for primitive data types
- **Escape Sequences –** `"\n"`, `"\t"`, etc all count as 1 character in `s.length()`
  - `"\\"` prints \ to the screen, because / is an escape sequence
  - `"//"` prints // to the screen
- **Casting** – if you cast a `double` into an `int`, you truncate the answer
- **Two parameter substring** - exclusive on second parameter
  - `String s = "Hello"; s.substring(1,3)` results in `"el"`
- **Short Circuit Evaluation** – only evaluate condition of an `if` when necessary
  - When part of `&&` is FALSE, don't continue to evaluate
  - When part of `||` is TRUE, don't continue to evaluate
- **Array vs ArrayList Syntax** – Array use `[],` ArrayList use `.get(i)`
- **Length vs Size** – Arrays use `.length`, Strings use `.length()`, ArrayLists use `.size()`
- **Nested Loops** – Always rows by columns

```
for(int r = 0; r < TwoDArray.length; r++)
        for(int c = 0; c < TwoDArray[0].length; c++)
            TwoDArray[r][c] = ...
```

- **Watch your Semicolons –** no ;s before `{}`'s, No ;s at the end of `if`, `for`, `while`, methods, etc
- **Removing in an ArrayList –** ArrayLists reindex, so if searching for something, you need an `i--;` after removing an element; alternatively, you can loop through backwards
- **Don't change list size in a for-each loop –** Use a regular for-loop instead in these cases.
- **Random Numbers –**

```
Random randGen = new Random();
int x = randGen.nextInt(10); //yields a # btwn 0–9
```

- **Recursion** – trace the recursion on paper (bubbles), do not do it all in your head!

Classes

- **Overriding methods** – make sure you name the method EXACTLY as it appears in the super class (parameters and return types included)
- **Inherited methods** – "`extends _____`", makes this class inherit all methods from _____
- **`this` vs `super`** – `this` refers to the class you are in, `super` refers to the class being extended
- **`instanceof` –** use to determine if an object is of a certain type
- **`private` vs `protected` vs `public`** – indicates which files that data can be accessed in
    - `private` – `this` class only
    - `protected` – `this` class and all subclasses
    - `public` – all classes
- **constructors** – `public`, no return type (not even void), same name as class
- **`public String toString()`** – returns a `String`, does NOT print anything!
- **`public int compareTo(Object x)`** – when "`implements Comparable`"
    - Cast parameter `x` to class type
    - `return this.var – other.var` for `int/double` OR
      `return this.var.compareTo(other.var)` for `String`
- **Polymorphism** – `SuperClass x = new SubClass();` a subclass can "act like" a super class, which means that here only methods that are in SuperClass may be called on `x`, but when those methods are invoked they will actually be called in the SubClass (unless the method doesn't exist there, then the inherited method from the SuperClass is called).

GridWorld

- **Critter methods –** Do not override the `act()` method!!
    - `public ArrayList<Actor> getActors()`
    - `public void processActors(ArrayList<Actor> actors)`
    - `public ArrayList<Location> getMoveLocations()`
    - `public Location selectMoveLocation(ArrayList<Location> locs)`
    - `public void makeMove(Location loc)`
- **Actor accessor methods –** `getGrid()`, `getColor()`, `getDirection()`, `getLocation()`
- **Grid methods**
    - `getGrid().isValid(loc)` – tells you if a location is valid in the grid
    - `getGrid().get(loc)` – returns actor at Location `loc` in the grid, or `null` if empty
- **Direction** is an `int`
    - `0 = Location.NORTH, 180 = Location.SOUTH, etc`
- **Location methods**
    - `new Location(0,1)` – creates a new Location with row 0, col 1
    - has `getRow()` and `getCol()` – you CANNOT call these in the actor class without calling `getLocation()` or calling them on another `Location` variable
    - `loc1.getDirectionToward(loc2)` – returns `int` direction from `loc1` to `loc2`
    - `loc3.getAdjacentLocation(dir)` – returns `Location` in direction `dir` from `loc3`