Review Cheat Sheet (CSE 142 Material)

Structure of a Java program (1.2)

```
public class name {
    public static void main(String[] args) {
        statements;
    }
}
```

System.out.println statement (1.2)

```
System.out.println("text");
System.out.println();        (blank line)
```

Comment (1.2) *(notes in program for coder)*

```
// text (one line)
/* text (multiple lines) */
```

Primitive types (2.1)

Type	Description	Examples
int	integers	42, -3, 92851
double	real numbers	3.14, 2.0
char	a character of text	'a', 'X', '\n'
boolean	logical values	true, false

Arithmetic Operators	
Operator	**Meaning**
+	addition
-	subtraction, negation
*	multiplication
/	division
%	remainder ("modulus")

Variables (2.2) *(pieces of memory that can store a value)*

```
type name;              declaration (a variable with no value)
name = value;           assignment (stores value into variable)
type name = value;      declaration/initialization (creates a variable and stores a value into it)
int y = 3;
int x = 1 + y * 2;      // x stores the value 7
```

The `for` loop (2.3) *(repeats a group of statements a fixed number of times)*

```
for (initialization; test; update) {
    statement;
    ...
    statement;
}
```

```
for (int line = 1; line <= 4; line++) {
    for (int j = 1; j <= (-1 * line + 4); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

```
...1
..2
.3
4
```

Class constants (2.4)

```
public static final type name = value;
public static final int DAYS_PER_WEEK = 7;
```

`Math` class (3.2) - *A set of useful methods for performing mathematical operations*

Method	Description	Method	Description
Math.abs(*value*)	absolute value	Math.max(*value1*, *value2*)	larger of two values
Math.ceil(*value*)	rounds up	Math.min(*value1*, *value2*)	smaller of two values
Math.cos(*value*)	cosine, in radians	Math.pow(*base*, *exponent*)	*base* to the *exponent* power
Math.floor(*value*)	rounds down	Math.round(*value*)	nearest whole number
Math.log(*value*)	logarithm, base *e*	Math.sin(*value*)	sine, in radians
Math.log10(*value*)	logarithm, base 10	Math.sqrt(*value*)	square root

```
double bigger = Math.max(Math.pow(2, 3), Math.sqrt(49));
System.out.println("The bigger value is " + bigger);
```

Constant	Description
Math.E	2.7182818...
Math.PI	3.1415926...

Parameters (3.1) *(A way to pass information in to a method)*

```
public static void name(type name, ..., type name) {
    statements;
}
methodName(expression, ..., expression);
```

Example:

```
public static void lineOfStars(int length) {
    for (int i = 1; i <= length; i++) {
        System.out.print("*");
    }
}
```

```
public static void main(String[] args) {
    lineOfStars(7);
}
```

Return (3.2)

(A way to pass information out from a method to its caller)

```
public static type name(parameters) {
    statement(s);
    ...
    return expression;
}
```

```
public static double fToC(double degreesF) {
    double dC = 5.0 / 9.0 * (degreesF - 32);
    return dC;
}
```

Scanner (3.4, 5.3)

(An object to read values from the keyboard)

```
import java.util.*;
Scanner console = new Scanner(System.in);
System.out.print("How old are you? ");
int age = console.nextInt();
System.out.println("You'll be 40 in " +
        (40 - age) + " years.");
```

Method	Description
nextInt()	reads/returns input as int
nextDouble()	reads/returns input as double
next()	reads/returns input as String

Method	Description
hasNextInt()	returns true if there is a next token and it can be read as an int
hasNextDouble()	returns true if there is a next token and it can be read as a double
hasNext()	returns true if there are any more tokens of input to read
hasNextLine()	returns true if there are any more lines of input to read

if/else statement(4.2)

(conditional execution)

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

Operator	Description
<	less than
<=	less than or equal
>	greater than
>=	greater or equal
==	equal
!=	not equal
&&	and
||	or
!	not

while loop (5.1)

(indefinite loops)

```
while (condition) {
    statement(s);
}
```

Example:

```
int number = 1;
while (number <= 200) {
    System.out.print(number + " ");
    number = number * 2;
}
```

boolean (5.2)

(logical values of **true** *or* **false**)

```
boolean minor = (age < 21);
if (minor) {
    System.out.println("Can't buy alcohol!");
}
public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```

Random (5.1)

(produce random numbers)

```
import java.util.*;   // for Random

// random value between 0 and 9
Random rand = new Random();
int randomNumber = rand.nextInt(10);
```

Method	Description
nextInt()	random integer
nextInt(**max**)	random integer between 0 and *max*
nextDouble()	random real# between 0.0 and 1.0
nextBoolean()	random true/false result

String (3.3, 4.4) - *store text*

```
String name = "P. Diddy";
System.out.println(name.length());   // 8
```

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

String Method	Description
`contains(`**str**`)`	true if this String contains the other's characters inside it
`endsWith(`**str**`)`, `startsWith(`**str**`)`	true if this String starts/ends with the other's characters
`equals(`**str**`)`	true if this String is the same as *str*
`equalsIgnoreCase(`**str**`)`	true if this String is the same as *str*, ignoring capitalization
`indexOf(`**str**`)`	index in this String where given String begins (-1 if not found)
`length()`	number of characters in this String
`substring(`**i**`, `**j**`)`	characters in this String from index *i* (inclusive) to *j* (exclusive)
`toLowerCase()`, `toUpperCase()`	a new String with all lowercase or uppercase letters

Reading an input file (6.1 - 6.3)

```
import java.io.*;    // for File
import java.util.*;  // for Scanner

public class name {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("file name"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            Scanner lineScan = new Scanner(line);
            process this line;
            ...
        }
    }
}
```

Example:

```
Scanner input = new Scanner(new File("input.txt"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    Scanner lineScan = new Scanner(line);
    int count = 0;
    while (lineScan.hasNext()) {  // count words on each line
        String word = lineScan.next();
        count++;
    }
    System.out.println("Line has " + count + " words");
}
```

Input cursor

```
308.2  14.9\n 7.4\n
^

input.nextDouble()
308.2  14.9\n 7.4\n
     ^

input.nextDouble()
308.2  14.9\n 7.4\n
           ^
```

Common Scanner errors

- `NoSuchElementException` - You read past the end of the input.
- `InputMismatchException` - You read the wrong type of token (e.g. read `"hi"` as `int`).

Field (8.1) *(data inside each object)*

```
private type name;
```

Method (8.2) *(behavior inside each object)*

```
public type name(parameters) {
    statements;
}
```

Inheritance (9.1 - 9.2)

```
public class name extends superclass {
```

Constructor (8.3) *(code to initialize new objects)*

```
public className(parameters) {
    statements;
}
```

`toString` method (8.6) *(called when an object is printed)*

```
public String toString() {
    code that produces/returns a String;
}
```

`super` keyword (9.3)

```
super.method(parameters)        // call an overridden method from superclass
super(parameters);              // call a constructor from superclass
```

Declaring and using arrays (7.1)

```
type[] name = new type[length];
name[index] = value;
```

Example:

```
int[] numbers = new int[10];
numbers[3] = 42;
numbers[7] = 23;
```

index	0	1	2	3	4	5	6	7	8	9
value	0	0	0	42	0	0	0	23	0	0

```
type[] name = {value, value, ..., value};
```

Example:

```
int[] numbers2 = {18, 7, 1, -3, 29, 4};
```

index	0	1	2	3	4	5
value	18	7	1	-3	29	4

Array as parameter (7.1)

```
public static void name(type[] name) {
```

Example:

```
public static double average(int[] nums) {
    int sum = 0;
    for (int i = 0; i < nums.length; i++) {
        sum += numbers[i];
    }
    return (double) sum / nums.length;
}
```

Array as return value (7.1)

```
public static type[] name(parameters) {
```

Example:

```
public static int[] countDigits(int n) {
    int[] counts = new int[10];
    while (n > 0) {
        counts[n % 10]++;
        n = n / 10;
    }
    return counts;
}
```

Array traversals (7.2)

```
for (int i = 0; i < array.length; i++) {
    do something with array[i];
    ...
}
```

Example:

```
int[] counts = {10, 30, 20, 4};
int sum = 0;
for (int i = 0; i < counts.length; i++) {
    sum += counts[i];
}
```

String traversals (4.4)

```
for (int i = 0; i < string.length(); i++) {
    do something with string.charAt(i);
    ...
}
```

Example:

```
String phrase = "the quick brown fox";
int capitalLetters = 0;
for (int i = 0; i < phrase.length(); i++) {
    char letter = phrase.charAt(i);
    if (letter >= 'A' && letter <= 'Z') {
        capitalLetters++;
    }
}
```

Methods of the `Arrays` class

Method	Description
`Arrays.copyOf(`**array**`, `**length**`)`	returns a new array with same elements as given array
`Arrays.equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays have the same elements
`Arrays.fill(`**array**`, `**value**`)`	sets every element in the array to the given value
`Arrays.sort(`**array**`)`	arranges the elements in the array into ascending order
`Arrays.toString(`**array**`)`	returns `String` for array, such as: `"[10, 30, 17]"`

```
System.out.println(Arrays.toString(numbers));  // [0, 0, 0, 42, 0, 0, 0, 23, 0, 0]
```

Common Array Errors

- `ArrayIndexOutOfBoundsException`
  You tried to access an element with an invalid index (a negative index, or ≥ the length of the array).