Java Cheat Sheet
2022 Hack the Ram
Intro to Object Oriented Programming (in Java)

- **Printing to console**

```
System.out.println("My message");
```

- **Comments**

```
//I am a comment, I am not ran by the code
```

- **Declaring/Instantiation (Variables)**

```
datatype varName = value;
int myVar = 5;
```

- **Boolean operators/if statements**
    - Boolean statement or value may be referred to as a condition.
    - && is AND operator - meaning both conditions must be true
    - || is OR operator - meaning one of the two conditions must be true
    - ! is NOT operator - meaning the opposite of condition it is in front of

```
int age1 = 19;

if(age1 < 20 && age1>12) // < 20 and > 12
    System.out.println("I am a teen");
else if(age1 == 18 || age1 == 19 ) // 18 or 19
    System.out.println("I am an adult teen");
else if(age1 > 19)
    System.out.println("I am an adult");
else  //anything else
{
    System.out.println("I am a child");
    if(age1 != 12)
        System.out.println("I am not a tween");
} // {} only needed if more than one statement!
```

Output:
I am an adult teen

Note: else if statement will only run when if statement is false

-**Lists:**

```
//create list
ArrayList<String> list1 = new ArrayList<String>();
list1.add("Hello"); //adds to list
list1.add("yooo");
list1.add(0,"noah"); //adds to 0th position
list1.add("yes");
list1.remove(2); //removes 2nd position
System.out.println(list1.get(1)); //prints 1st position
System.out.println(list1); //prints the list
```

- **While loop**

Ex:

```
while(condition/boolean [ex: number > 6])
{
        //Code inside block
}
```

- **For loop**

```
for(int var = value; condition [ex: var < 3]; modify variable [ex: var++] )
{
        //Code inside block
}
```

Output:
Num: 64
Num: 16
Num: 4

Ex:

```
//for loops execute code in the block {} while
//the condition (in the middle) is met

//for loops are while loops, that keep track and modify
//a set variable (commonly i) each iteration (loop)

//so this loop
//repeats WHILE i is less than 5
for(int i = 0; i<5; i++) //i++ increments i by 1
{
    System.out.println("Hello, " + i);
    //prints out Hello, [current value of i]
    //then adds 1 to i
}
```

Output:
Hello, 0
Hello, 1
Hello, 2
Hello, 3
Hello, 4

- **Enhanced For Loop**

Assume myList is an ArrayList (see Lists section above)

```
for(String word : myList)
{
    System.out.println(word);
}
```

- **Methods**
    - can be called to perform function
    - part of class
    - contain return type
    - can call methods by name followed by parentheses.
        - myMethod()

```
public static returntype methodName(datatype argument)
{
        //code here
        return [can return whatever return type specified, see examples below]
}
```

Ex:

```
public static void main(String[] args)
{
    beCute();
    int product = sqrt(5);
    System.out.println(product + ", " + sqrt(product));
}

public static void beCute()
{
    System.out.println("^-^");
}

public static int sqrt(int x)
{
    return x * x;
}
```

Method call

Use void return type if returning nothing

Return type

Output:
^-^
25, 625

- **<u>Classes</u>**
- Can be runnable (if they contain main method) [Driver Class]
- May just contain useful methods (functions) [Utility/Helper Class]
- Can be used as a blueprint to create objects

- Calling static methods from other classes

ClassName.*method*();
For example:

Given this class:

```
public class CoolMethods
{
    public static int power(int number, int power)
    {
        int newNum = 1;
        for(int i = 0; i<power; i++) //while i is less than power
        {
            newNum*=number;
        } //increments i by 1
        return newNum;
    }

    public static boolean startsWithN(String word)
    {
        word = word.toLowerCase(); //sets word to be lowercase
        if(word.substring(0,1).equals("n"))
            return true;
        return false;
    }
}
```

Call these methods like this:

```
public class OtherClass
{

    public static void main(String[] args)
    {
        int num = CoolMethods.power(2, 3);
        System.out.println(num);

        String name = "Noah";
        System.out.println(CoolMethods.startsWithN(name));

    }

}
```

Creating an object:

Classname varName = **new** Classname(parameters);

Calling object method (non-static):

varName.method();

Ex:

```
Thingy gadget = new Thingy(5);
gadget.doSomething();
```

"gadget does something"

Writing class:
Fields - Global variables to the class
Methods - functions each object can run
- Independent of one another

Overloading - Two methods containing the same name with different parameters

Class example:

```
public class Line
{
    private int length; //field

    //constructor
    //called to create object
    public Line(int l)
    {
        length = l;
    }

    //returns length field
    // "getter"
    public int getLength()
    {
        return length;
    }

    //sets length field
    // "setter"
    public void setLength(int l)
    {
        length = l;
    }

    //this is a method
    public void increaseLength()
    {
        length++;
    }

    //overloaded method
    public void increaseLength(int num)
    {
        length += num;
    }

}
```

Example use:

```
public class OtherClass
{
    public static void main(String[] args)
    {
        Line arm = new Line(5);
        arm.increaseLength();
        arm.increaseLength(2);
        System.out.println(arm.getLength());
    }
}
```

This class displays what every Line object will have and be able to do.
Inheritance
- Use <u>extends</u> keyword to inherit from another class
- child class inherits all parent class public methods/fields
- code reuse

Ex:

```
public class SlantedLine extends Line  //parent class: Line
{
    private double angle; //field

    public SlantedLine(int length, int an)
    {
        super(length); //calls super class constructor
        angle = an;
    }

    public double getAngle()
    {
        return angle;
    }

    public void turnRight()
    {
        angle+=90;
    }
}
```