Programming concepts

A programming paradigm is a way of conceptualizing what it means to perform computation, and how tasks that are to be carried out on a computer should be structured and organized.

A program has two components data and algorithms.
0.5cm

Imperative paradigm: Also know as algorithmic paradigm, think C. No classes, references to memory, sequentiel running of program/algorithm.

Procedural paradigm: Same as Imperative except it's more modular, meaning code is easily reusable.

Declarative paradig: Computer finds the solution to a problem, think SQL. The data is "permanent".

Functional paradigm: Based on mathematical functions. Data is immutable (Underlying structure cannot be changed). Java allows for lambda functions which allows for creating functional programming.

Logic paradigm: Set the goal and specify the problem, to make the computer solve the problem, instead of writing an algorithm for finding said solution.

Object-oriented par: Objects contains the data and algorithms. Where data is the state of the object. The algorithms allows to change the state of an object. Objects are often described as classes in programming. Think Java, C# etc.

Polymorphism

Overloading Polymorphism

Multiple definition of a function with varying parameters.

```
public class MathUtil {
    public static int max(int n1, int n2) {
        /*code*/
    }
    public static double max(double n1, double n2) {
        /*code*/
    }
    public static int max(int[] num) {
        /*code*/
    }
}
```

Coercion Polymorphism

When a type is implicitly converted.

```
int num = 707;
double d1 = (double)num; // Explicit
double d2 = num; // Implicit
```

Inclusion Polymorphism

Subtype, meaning a type/class that extends or implements a type/class can also be assigned to it's parent type. Think employee extends person can also be assigned to a variable with type person.

Parametric Polymorphism

Lets classes and algorithms be generic without type specification

```
List<String> sList = new ArrayList<String>();

public <T> T example(T test) {
    return test;
}

class test<T> {
    /*code*/
}
```