The Purpose of Methods

Methods act as an interface between a program and the data fields of a class in the program.

These methods can either alter the content of the data fields or use their values to perform a certain computation. All the useful methods should be public, although, some methods which do not need to be accessed from the outside could be kept private.

Definition and Declaration#

A method is a group of statements that performs some operations and may or may not return a result.

Here is an example of a method in Java:

```
class Car {
     // Public method to print speed
      public void printSpeed(int speed) {
        System.out.println("Speed: " + speed);
9
    class Demo {
10
11
      public static void main(String args[]) {
12
13
       Car car = new Car();
       car.printSpeed(100); // calling public method
14
15
16
17
                                                                                                   Reset
                                                                                          Save
```

Method parameters make it possible to pass values to the method and return type makes it possible to get

Method Parameters, Return Type

the value from the method. The parameters are declared inside the parentheses after the method name while the return type is declared before method name. Return Statement

For methods that define a return type, the **return statement** must be immediately followed by return value.

// public method with one parameter speed.

```
public int printSpeed(int speed) {
   // ...
   return speed + 5; // return statement
This method adds 5 to the parameter speed passed to it and returns the result.
```

The return type, int, which comes before the method name **PrintSpeed**, indicates that this method returns an int.

Getters and Setters

These two types of methods are very popular in OOP. A **get** method retrieves the value of a particular data field, whereas a set method sets its value.

private int speed; // member field speed

// Car class

class Car {

It is a common convention to write the name of the corresponding member fields with the get or set command.

Let's write get and set methods for the 'speed' in our Car class:

```
5
        // Setter method to set the speed of the car
   6
        public void setSpeed(int x) {
          speed = x;
  10
        // Getter method to get the speed of the car
  11
        public int getSpeed() {
  12
          return speed;
  13
        }
  14
  15
  16
  17
      class Demo {
  18
  19
         public static void main(String args[]) {
  20
           Car car = new Car();
  21
           car.setSpeed(100); // calling the setter method
  22
           System.out.println(car.getSpeed()); // calling the getter method
  23
  24
  25
  26
                                                                                                 Reset
   Run
Method Overloading
```

Methods can be overloaded in Java. We could redefine a method several times and give it different arguments and method types. When the

return x * y;

return x * y * z;

arguments.

Overloading refers to making a method perform different operations based on the nature of its

Let's see this in action by overloading the product method in the Calculator class: class Calculator {

method is called, the appropriate definition will be selected by the compiler!

public double product(double x, double y) {

// Overloading the function to handle three arguments

public double product(double x, double y, double z) {

Advantages of Method Overloading

don't have to keep track of different methods.

```
10
   11
         // Overloading the function to handle int
   12
         public int product(int x, int y){
   13
   14
           return x * y;
         }
   15
   16
   17
   18
       class Demo {
   19
   20
   21
         public static void main(String args[]) {
           Calculator cal = new Calculator();
   22
   23
           double x = 10;
   24
           double y = 20;
   25
           double z = 5;
   26
   27
   28
           int a = 12:
   Run
                                                                                                    Reset
In the code above, we see the same method behaving differently when encountering different types of inputs.
    Note: Methods which have no arguments and differ only in the return types cannot be overloaded
    since the compiler won't be able to differentiate between their calls.
```

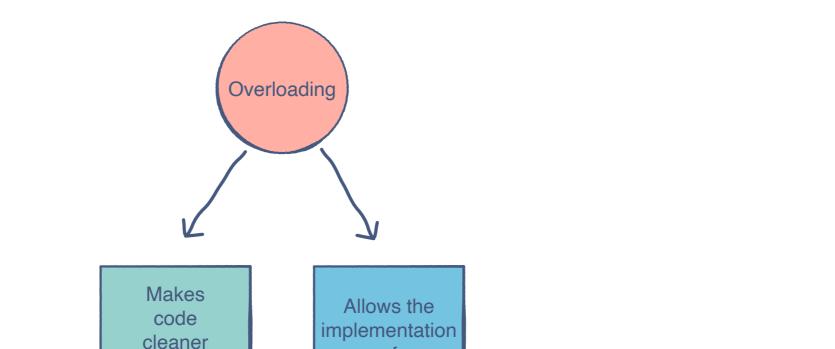
One might wonder that we could simply create new methods to perform different jobs rather than

overloading the same method. However, an obvious benefit is that the code becomes simple and clean. We

Polymorphism is a very important concept in object-oriented programming. It will come up later on in the course, but method overloading plays a vital role in its implementation.

and

readable



polymorphism