**메소드 기초**

**package javastory.club.stage1.step1;**

**public class StoryAssistant {**

**//**

**public static void main(String[] args) {**

**//**

***createTravelClub();***

**}**

**public static void createTravelClub() {**

**//**

**String name = "JavaTravelClub";**

**String intro = "Travel club to Java island.";**

**TravelClub club = new TravelClub(name, intro);**

**System.*out.println(club.toString());***

**}**

**}**

메소드는 클래스 안에 위치합니다. StoryAssistant 클래스 안에는 main 메소드와 createTravelClub 메소드 두 개가 존재합니다. 메인 메소드의 첫 문장은 createTravelClub 메소드를 호출합니다. 호출된 메소드는 위에서부터 차례대로 문장을 수행한 뒤 종료됩니다. 위 코드는 결국에는 아래 코드와 같은 결과를 갖습니다.

**package javastory.club.stage1.step1;**

**public class StoryAssistant {**

**//**

**public static void main(String[] args) {**

**//**

**String name = "JavaTravelClub";**

**String intro = "Travel club to Java island.";**

**TravelClub club = new TravelClub(name, intro);**

**System.*out.println(club.toString());***

**}**

**}**

아래 코드가 더 간결해 보입니다. 그런데 굳이 왜 위의 코드처럼 메소드를 사용하였을까요? 바로 여러 문장의 코드를 하나의 행동으로 묶고, 필요할 경우 코드를 재사용하기 위해서입니다. 메소드를 사용한 위의 코드와 메소드를 사용하지 않은 아래 코드를 비교해보면 그 이유를 바로 알 수 있습니다. 아래의 코드처럼 별도의 메소드를 만들지 않았을 때 메인 메소드 내의 4개의 문장이 의미하는 바를 이해하기 위해서는 한 줄 한 줄 코드를 해석해서 유추하는 수밖에 없습니다. 하지만 메소드로 코드를 묶으면 코드를 이해하기 훨씬 쉽습니다. 위 코드의 createTravelClub 메소드의 이름만으로 우리는 이 메소드가 TravelClub을 생성한다는 것을 유추할 수 있습니다.

메소드는 코드의 재사용에도 큰 도움을 줍니다. 코드의 재사용에 용이하다는 사실은 쉽게 유추하실 수 있을 것입니다. 위 코드에서 2개의 TravelClub을 생성한다고 생각해 보십시오. 메소드가 없는 경우에는 4줄을 더 작성해야 합니다. 하지만 메소드가 존재할 경우 메소드를 한 번 더 호출하기만 하면 됩니다.

**(참고 : 메소드의 네이밍)**

* 메소드는 하나의 행동을 나타냅니다. 따라서 메소드의 이름은 그 메소드가 하는 일을 잘 반영하여야 합니다. 똑같이 작동한다고 메소드의 이름을 아무렇게나 지어버리면 코드가 조금만 커져도 알아보기 힘듭니다. 따라서 메소드의 이름을 지을 때에는 메소드의 이름이 행동을 잘 반영하는지 (생각보다 더) 신중하게 고민하여야 합니다.