A *data model*, sometimes also called *domain model*, represents the data you want to work with. For example, if you develop an online flight booking application, you might model your domain model with objects like Person, Flight, Booking etc. The EMF tooling allows you to create UML diagrams.

A good practice is to model the data model of an application independently of the application logic or user interface. This approach leads to classes with almost no logic and a lot of properties, e.g., a Person class could have the firstName, lastName, Address properties, etc.

With EMF you define your domain model explicitly. This helps to provide clear visibility of the model. The code generator for EMF models can be adjusted and in its default setting. It provides change notification functionality to the model in case of model changes. EMF generates interfaces and a factory to create your objects; therefore, it helps you to keep your application clean from the individual implementation classes.

Another advantage is that you can regenerate the Java code from the model at any point in time.

[**1.2. Eclipse Modeling Framework (EMF)**](https://www.vogella.com/tutorials/EclipseEMF/article.html#eclipse-modeling-framework-emf)

The *Eclipse Modeling Framework* (EMF) is a set of Eclipse plug-ins which can be used to model a data model and to generated code or other output based on this mode. EMF has a distinction between the meta-model and the actual model. The meta-model describes the structure of the model. A model is a concrete instance of this meta-model.

EMF allows the developer to create the meta-model via different means, e.g., XMI, Java annotations, UML or an XML scheme. It also allows to persists the model data; the default implementation uses a data format called *XML Metadata Interchange*.

[**1.3. Generate data from an EMF model**](https://www.vogella.com/tutorials/EclipseEMF/article.html#generate-data-from-an-emf-model)

The information stored in the EMF models can be used to generate derived output. A typical use case is that you use EMF to define the domain model of your application and that you generate the corresponding Java implementation classes from this model. The EMF framework supports that the generated code can be safely extended by hand.

The EMF model (which holds real data based on the model structure) can also be used to generate different output, e.g., HTML pages, or it can be interpreted at runtime within an application.

[**1.4. Meta models - Ecore and Genmodel**](https://www.vogella.com/tutorials/EclipseEMF/article.html#meta-models-ecore-and-genmodel)

The EMF meta-model consists of two parts; the *ecore* and the *genmodel* description files.

The *ecore* file contains the information about the defined classes. The *genmodel* file contains additional information for the code generation, e.g., the path and file information. The *genmodel* file also contains the control parameter how the code should be generated.

[**1.5. Ecore description file**](https://www.vogella.com/tutorials/EclipseEMF/article.html#ecore-description-file)

The *ecore* file allows to define the following elements.

* EClass: represents a class, with zero or more attributes and zero or more references.
* EAttribute: represents an attribute which has a name and a type.
* EReference: represents one end of an association between two classes. It has flags to indicate if it represents a containment and a reference class to which it points.
* EDataType: represents the type of an attribute, e.g., int, float or java.util.Date

The *Ecore* model shows a root object representing the whole model. This model has children which represent the packages, whose children represent the classes, while the children of the classes represent the attributes of these classes.