Modul Panduan Raptor - Object Oriented Programming

Yang Tidak Dipublikasikan

Yusuf Lestanto, ST., MSc., MBA

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK DAN ILMU KOMPUTER
UNIVERSITAS BAKRIE
Semester Genap 2023/2024

LEMBAR PENGESAHAN

1. Judul PkM	:	Modul Panduan Raptor - Object Oriented Programming
2. Ketua Tim Pengabdian		
a. Nama lengkap	:	Yusuf Lestanto, ST., MSc., MBA.
b. NIDN	:	0302057105
c. Pangkat/Golongan	:	Penata Muda Tk. I - III/b
d. Jabatan	:	Dosen Tetap
e. Telp/Alamat Surel	:	087775084255 / yusuf.lestanto@bakrie.ac.id
3. Anggota Tim Pengabdian		
a. Dosen	:	1. .....
	:	2. .....
b. Praktisi	:	1. .....
	:	2. .....
4. Peserta		
a. Mahasiswa	:	1. .....
	:	2. .....
b. Alumni	:	1. .....
	:	2. .....
5. Biaya Kegiatan		
a. Universitas Bakrie	:	Rp. 0,-
b. Sumber lain	:	Rp. 0,-
6. Tahun Pelaksanaan	:	Tahun 2024

Jakarta, 3 Mei 2024

Mengetahui,
Kaprodi

Iwan Adhicandra, Ph.D. (Sydney), SMIEEE
NIP: 0018025806

Pelaksana Pengabdian

Yusuf Lestanto, S.T., MSc., MBA
NIDN: 0302057105

Mengetahui,
Ketua LPkM Universitas Bakrie

Prof. Ardiansyah, S.T.P., M.Si., Ph.D.
NIDN: 0318107501

Guidance Module: Raptor - Object Oriented Programming

Yusuf Lestanto

1 Basic Concepts of OOP

Object Oriented Programming (OOP) is a programming paradigm that focuses on creating objects that have attributes (data/variables) and methods (functions/procedures) to manipulate these data. OOP consists of four main concepts:

- Encapsulation

  Encapsulation bundles the data (attributes) and the methods (functions) that operate on the data into a single unit, called an **object**. It helps to control access to the data, protecting it from outside interference and misuse. The main characteristics are:

  - Data Hiding:

    By using access specifiers like **private**, **protected**, or **public**, they restrict or control how attributes and methods are accessed or modified from outside the class. By marking a variable as **private**, it means that variable cannot be accessed directly outside the class.

  - Getter and Setter Methods:

    Encapsulation is often implemented using **getter** and **setter** methods, which provide controlled access to the private data. This ensures that the data can only be accessed or modified in a safe and controlled manner.

  - Modularity:

    Encapsulation helps to achieve modularity by keeping the implementation details hidden. External classes can interact with the object through a well-defined interface without knowing the inner algorithms.

- Inheritance

  Inheritance allows a new class (called a child class or subclass) to inherit the attributes and methods of an existing class (called a parent class or superclass). Therefore, it enables code reusability and establishes a natural hierarchical relationship between classes. The main characteristics are:

  - Code Reusability:

    Inheritance allows the child class to reuse code and properties of the parent class, which helps in reducing code duplication.

- Hierarchical Structure:

  It models relationships like "is-a" or "kind-of." For example, if **Animal** is a parent class, a **Dog** class can be a child class, inheriting behaviors of the **Animal** class, because a *dog* is a type of *animal*.

- Extending Functionality:

  A child class can add its own unique attributes and methods or override (redefine) methods from the parent class to alter behavior as needed.

Inheritance has the following types:

- Single Inheritance:

  A child class inherits from one parent class.

- Multiple Inheritance:

  A child class inherits from more than one parent class (not supported directly in some languages like Java).

- Multilevel Inheritance:

  A child class is derived from another child class, creating a chain of inheritance.

- Hierarchical Inheritance:

  Multiple child classes inherit from the same parent class.

- Polymorphism

  The term "polymorphism" comes from the Greek words "poly" (meaning many) and "morph" (meaning forms). In the context of programming, polymorphism refers to the ability of different objects to respond to the same operation or method call in different ways. The essential features of Polymorphism are:

  - Method Overriding (Runtime Polymorphism):
    - Occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.
    - The method name and parameters are the same, but the behavior is customized in the subclass.
    - Used to achieve dynamic (or late) binding, where the method to be executed is determined at runtime.
  - Method Overloading (Compile-Time Polymorphism):
    - Occurs when multiple methods have the same name but different parameters (different type, number, or order of parameters).
    - Used to implement similar operations with variations in input.
    - The decision about which method to call is made at compile time based on the method signature.

Polymorphism allows programmers to design more flexible and easily maintainable code by enabling a single interface or method to represent multiple types or behaviors. This abstraction is particularly powerful when combined with inheritance and interfaces, as it enables objects to be treated as instances of their parent class while behaving according to their specific implementations.

- Abstraction

  Abstraction refers to the process of hiding the complex implementation details of an object and only exposing the essential features that are necessary for the user. Abstraction allows programmers to focus on what an object does rather than how it does it. The main features are:

  - Simplification:

    Abstraction simplifies code by reducing complexity. It allows you to interact with an object through a simplified interface without needing to know its inner workings.

  - Hiding Implementation:

    The internal processes and data handling are hidden from the outside world. For example, you may know that a **Car** object has a method *start()*, but you don't need to understand the specific mechanism of how the engine starts internally.

  - Interface-Driven Design:

    Abstraction often utilizes interfaces or abstract classes in languages like Java or C#, which define methods but do not implement them. Concrete classes that implement these interfaces provide the actual functionality.

2 Object-Oriented Mode

RAPTOR enables the development of basic object-oriented programs through classes, which contain attributes and methods. While it allows for the creation and instantiation of objects, it is essential to acknowledge that this only provides a superficial understanding of the vast and intricate subject of object-oriented programming.

To employ RAPTOR in an object-oriented programming (OOP) context, it is necessary to choose the Object-oriented mode, as depicted in Figure 1. There are two tabs available: **UML** and **main**. RAPTOR utilizes a form of UML to design the structure of an object-oriented program. Classes are defined in the UML screen by clicking on the UML tab. The button for adding a new class (highlight number 1) is illustrated in Figure 2. Additionally, a new **Return** symbol (highlight number 2) has been introduced to the set of symbols.

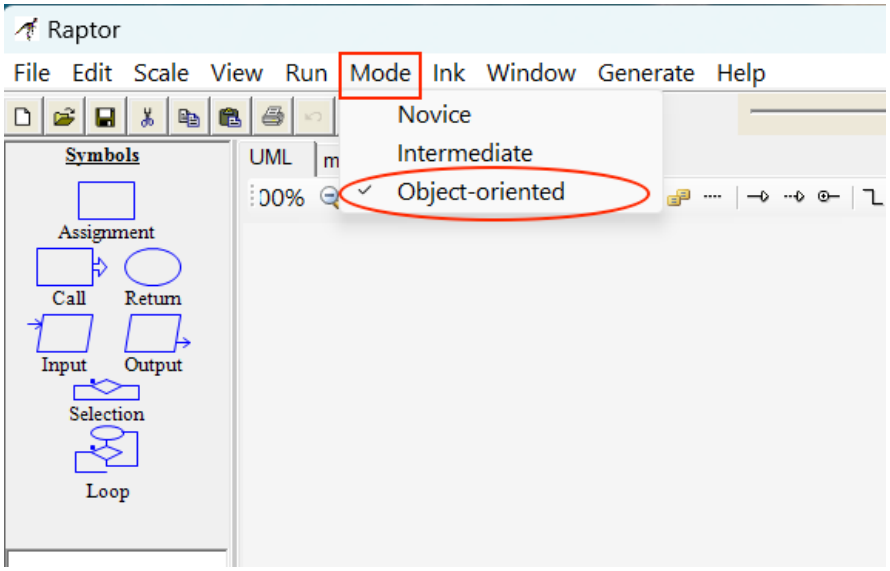

Figure 1: Set to OOP mode

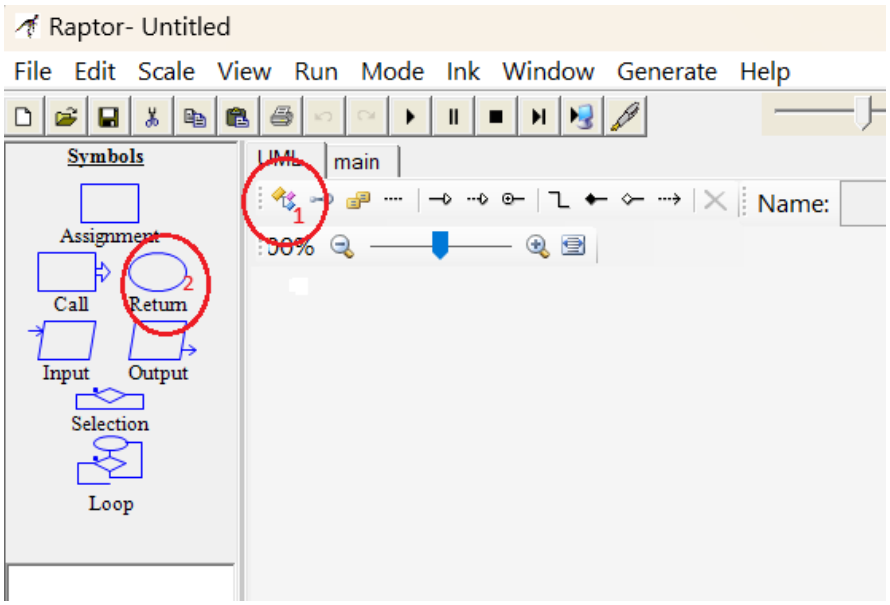

Figure 2: UML

3 Creating a Class

When the **Add New Class** button is selected then a new class is created, a Name field will be displayed. Provide a name for the class, as illustrated in Figure 3.

As shown in Figure 3, a class named **Car** has been created. To add members (methods and attributes), double-click inside the **Car** class. In RAPTOR, attributes are referred to as Fields. A new window will open, the class members are enabled to be entered (refer to Figure 4).

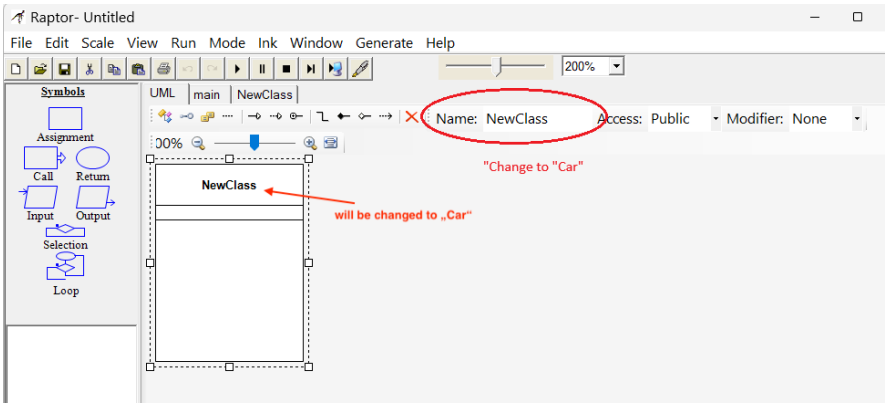

Figure 3: Creating a new class

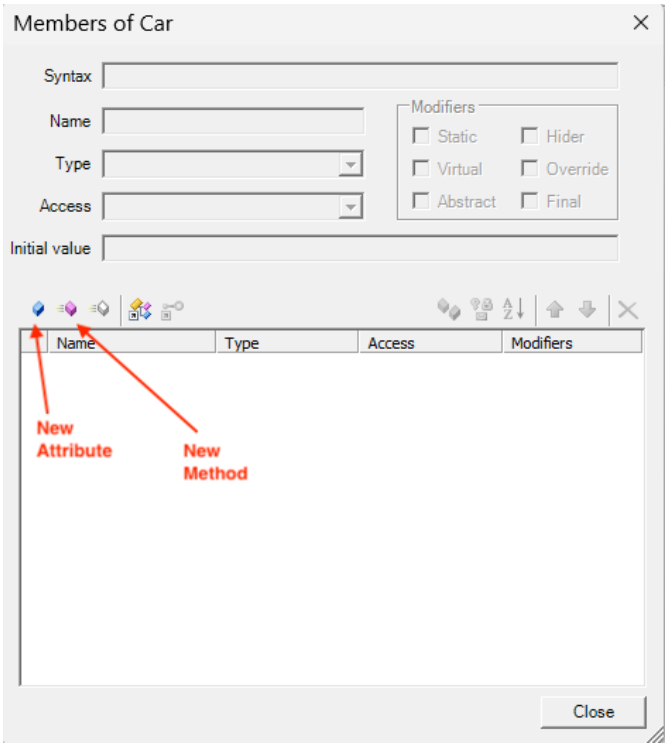

Figure 4: Adding members to a Class

3.1 Using the Car Class to Find the Production Year of a Car

A class named Car will be used to get the information of production year. For this purpose the following members will be needed:

- attribute: year (a number)
- methods: setYear(), getYear().

The Figure 5 shows the Class Car and its members.

- Note the syntax for a **Field**: A **Field** must be given a data type. The type of year is **int**.

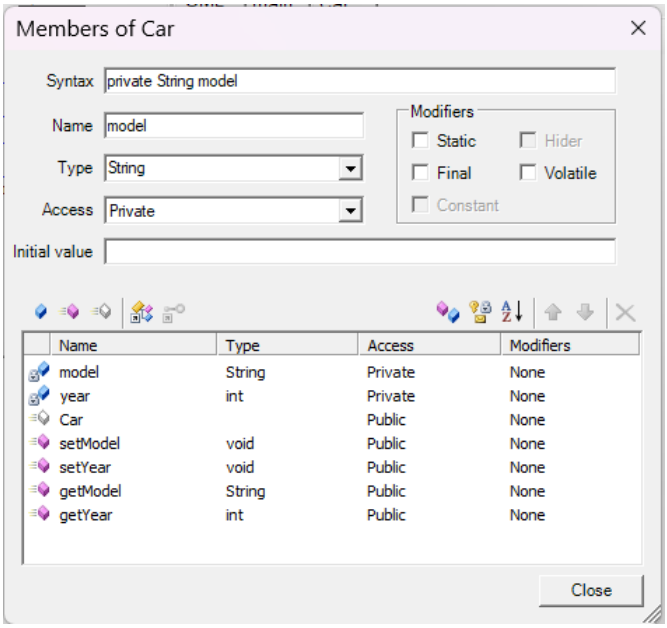

Figure 5: Class Car and its members

- Note the syntax for a **Method**. If the **Method** receives a value passed from **main**, the passing parameter must be included. For example,
  - The **Method setYear()** is passed a value for the year of a car so the syntax for this Method is

    public void setYear(int year)
  - The **Method setModel()** is passed a text for the model of a car so the syntax for this Method is

    public void setModel(String m)
  - The **Method getYear()** retrieves the value of the year of a car. Syntax for this Method is:

    public int getYear()
  - The **Method getModel()** retrieves the text of the model of a car. Syntax for this Method is:

    public String getModel()

Once the Class has been created, a new tab is automatically added, with the name of the Class (see Figure 6). Now the code for each of the Class's methods must be created. Click the Car tab to see five new tabs—one for each Method, as shown in Figure 7.

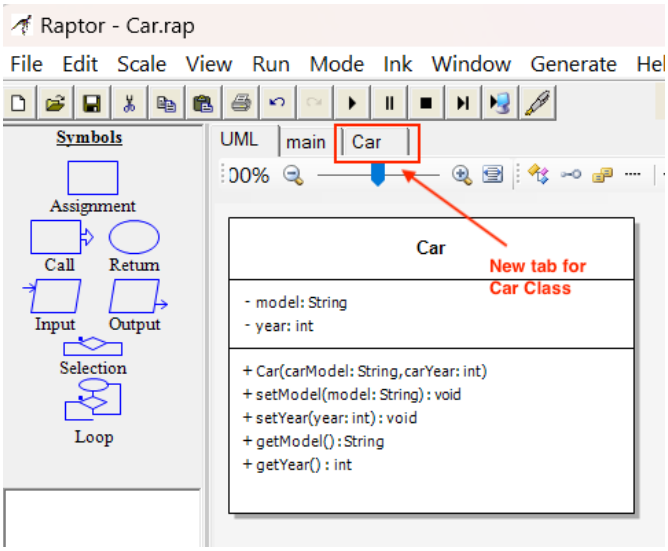

Figure 6: New tab for the Class Car

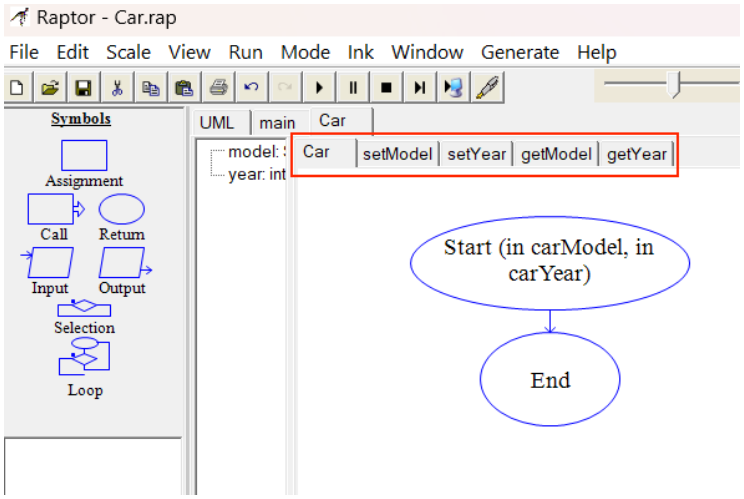

Figure 7: New tabs for each new Method

3.2 Code the Methods

In **Car** class, there is a **constructor** as a special method which is used to initialize objects when a class is instantiated (see Figure 8). It sets up the initial state of the object by assigning values to its attributes and performing any setup tasks required before the object can be used. The main characteritics of constructors:

- Same Name as Class

  A constructor has the same name as the class in most programming languages.

- No Return Type

  Constructors do not have a return type, not even **void**.

- Automatic Invocation

The constructor is called automatically when an object is created using the **new** keyword (or equivalent syntax).

- Parameterization

  Constructors can be parameterized to initialize the object with custom values, or they can be a **default constructor** with no parameters.

In above example, the **Car** class has a constructor that initializes the *model* and *year* and *make* attributes when a **Car** object is created. There are two types of constructors:

1. Default Constructor

   Takes no parameters and initializes objects with default values.

2. Parameterized Constructor

   Takes arguments to set specific values for the attributes.

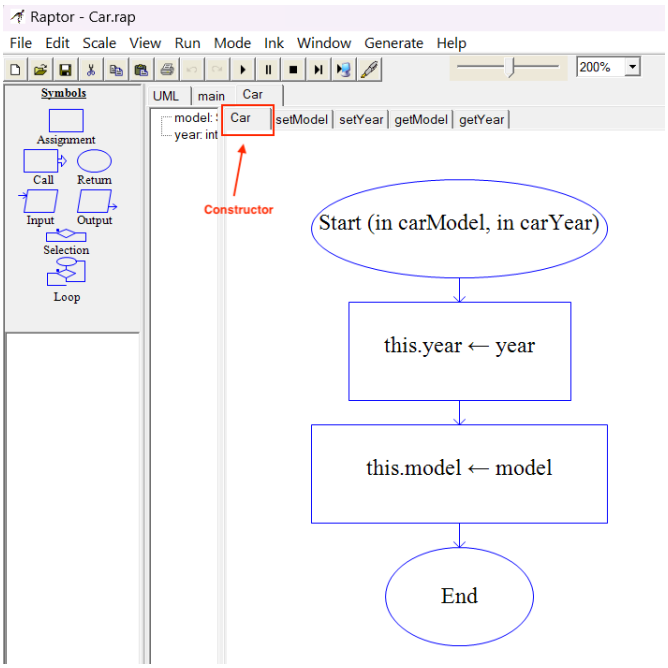

Figure 8: The Constructor Method

Figure 8 illustrates that this Car class is defined with an additional constructor that accepts parameters. When an object is created, the appropriate constructor will be invoked. This means that if a new object is instantiated without parameters, the default constructor will be executed.

The other methods for this **Car** are as follows: **setYear(year)**, **setModel(model)**, **getYear()**, **getModel()**.

3.2.1 Method setYear(year):

The setYear() Method does one thing only. It sets the value of the car's year, as passed to it from the main program, to the variable year. This assignment is done using the this keyword. The code for this method is shown in Figure 9.

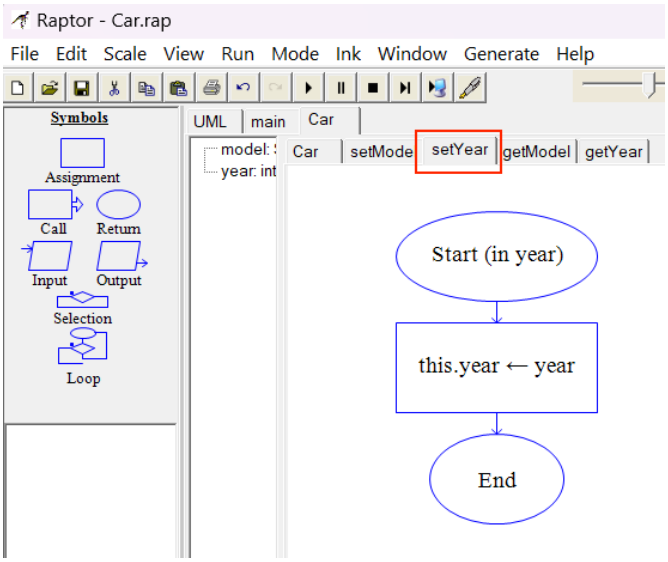

Figure 9: Code for the setYear() method

3.2.2 Method setModel(model):

The setModel() method has a single purpose: it assigns the car's model value, provided by the main program, to the model variable. This assignment is performed using the this keyword. Figure 10 displays the code for this method.

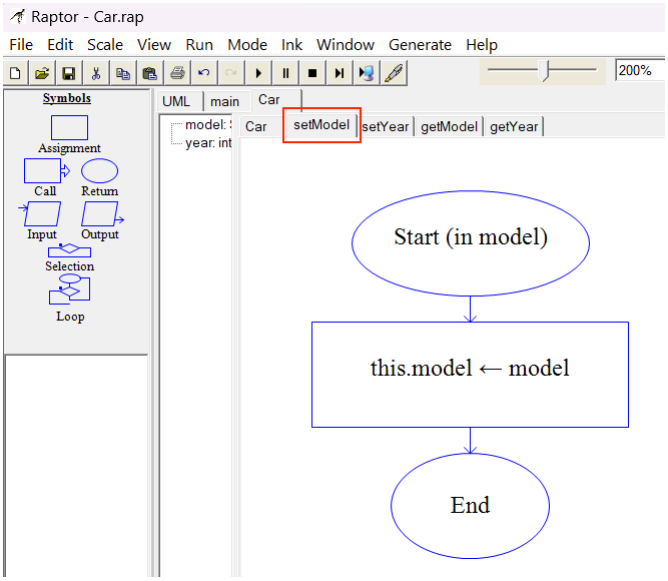

Figure 10: Code for the setModel() method

3.2.3 Method getYear():

The getYear() method retrieves the year value when invoked and returns it, as illustrated in Figure 11.

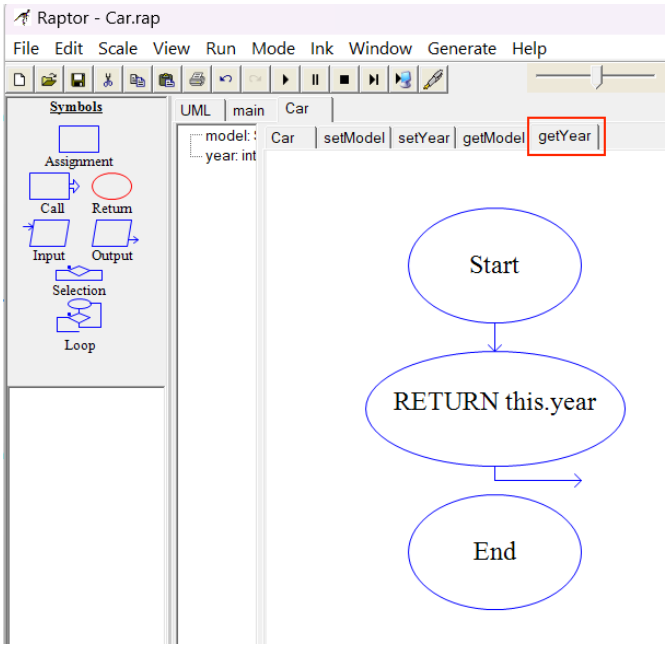

Figure 11: Code for the getYear() method

3.2.4 Method getModel():

The getModel() method is used to obtain the model's text when called, and it returns the text model, as depicted in Figure 12.

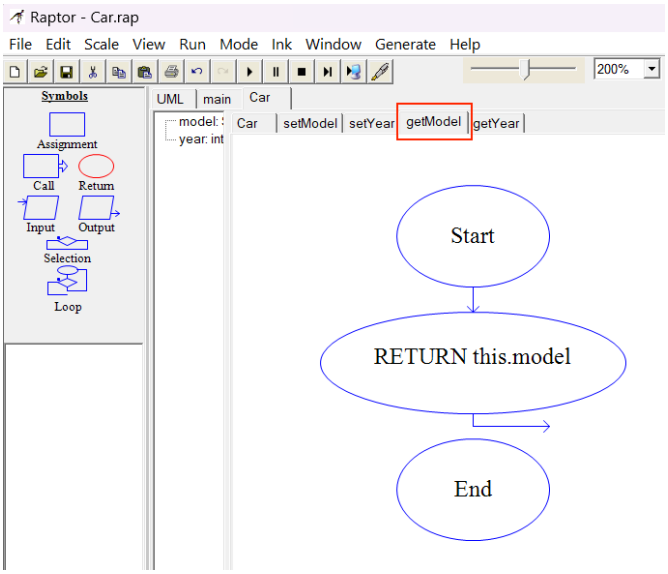

Figure 12: Code for the getModel() method

4 The main Program

Now we can create the Main program. This example is very straightforward; it enables the user to input a car's year and model, and then displays the entered information. This is achieved by creating an instance of the 'Car' class using the default constructor, and then utilizing the class's methods and attributes. Figure 13 illustrates how this is implemented in the RAPTOR OOP approach.

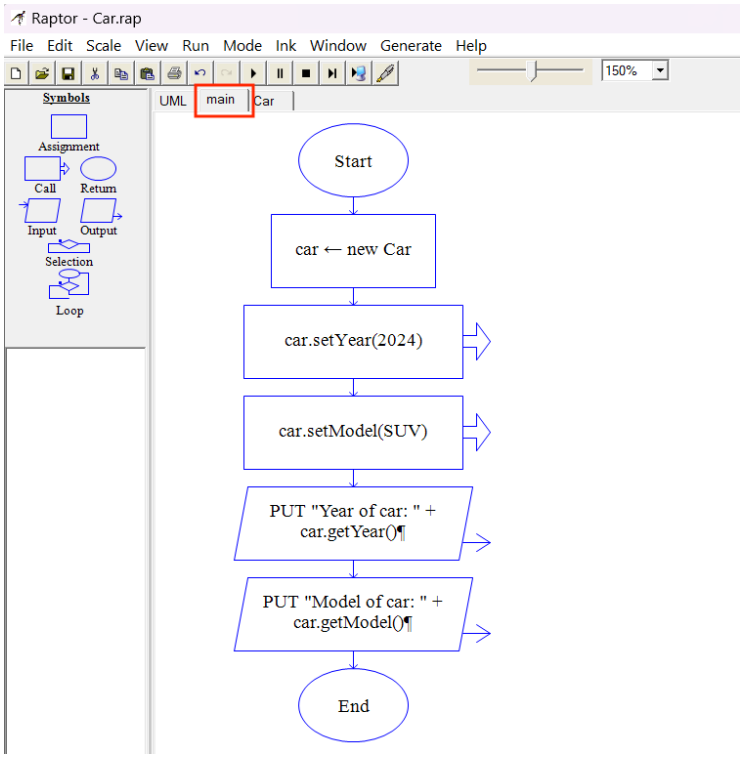

Figure 13: Code for the main() method

References

[1] Venit, S. Extended Prelude to Programming: Concepts and Design-with CD. (Scott/Jones Inc.,2003)

[2] Gaddis, T. Starting Out with Programming Logic and Design, 4/e. (Pearson Education,2015)