7 Lab: modelação, visualização e geração de código

Tópicos

Um modelo de interação mostra os participantes e a forma como colaboram para realizar um caso de utilização/função no sistema. Um participante solicita a colaboração de outro através do envio de mensagens. A UML fornece quatro diagramas para representar interações, sendo o mais prevalente o Diagrama de Sequência, que inclui um eixo temporal implícito. A semântica do diagrama de sequência é particularmente útil para representar a colaboração entre objetos , em Java.

Tarefas

E7.1

Considere o trecho de código em Java apresentado no Quadro 1.

Modele a colaboração entre objetos que decorre na execução do método ServicoTransferencias.transferir() , utilizando um diagrama de sequência.

Quadro 1

```
public class ServicoTransferencias {
    // base de dados com a informação dos clientes e contas
    private RepositorioContas repositorio;

    public void transferir(String idOrdenante, String idBeneficiario,
             double quantia) {
       ContaCorrente origem = repositorio.procurarContaDeCliente(idOrdenante);
       ContaCorrente destino =
repositorio.procurarContaDeCliente(idBeneficiario);

       origem.debitar(quantia);
       destino.creditar(quantia);
       repositorio.atualizarConta(origem);
       repositorio.atualizarConta(destino);
    }
...
}
```

E7.2

Nos próximos exercícios, vamos usar os diagramas da UML para criar visualizações do código em Java.

Para isso, vamos usar especificamente a integração VisualParadigm / Eclipse. Antes de avançar, propriamente dito, experimente usar o VisualParadigm para gerar código Java conforme explicado a seguir.

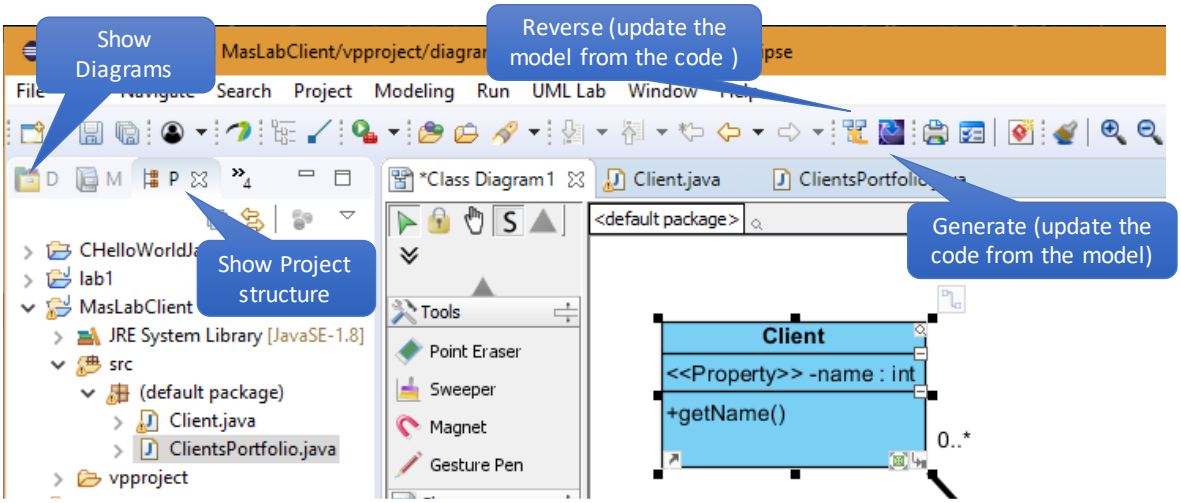
Modelação e geração de código Java com VisualParadigm e Eclipse:

Nos seguintes pontos são referidos dois tutoriais da documentação do VisualParadigm que explicam o processo de integração.

Não é para fazer tudo no primeiro; apenas o que está referido.

a) Aceda ao Tutorial 1 (Getting Started). Execute os passos sequencialmente até à secção "Generate Java Code from UML Class"; não faça a secção "Perform Coding" e seguintes. Note que depois de ter feito a integração (i.e., ficheiros copiados), deve fechar o VisualParadigm e trabalhar a partir do Eclipse).

b) Aceda ao Tutorial 2 (...Round-trip Engineering). Execute todos os passos sequencialmente. Note a referência no último ponto ("3. This is the end of the tutorial...") em que se sugere que introduza código nas classes e verifique que as alterações são refletidas no modelo.



E7.3

Considere os excertos de código disponíveis nos Quadro 2, Quadro 3 e Quadro 4 (adiante).

Utilizando um projeto Java no Eclipse, modele num diagrama de classes a informação estrutural que se pode depreender do código (não é preciso implementar os métodos).

Comece com um diagrama vazio e adicione os elementos visuais, passo a passo. (O código deve ir sendo gerado pela ferramenta, tanto quanto possível, e não escrito ou copiado.)

Nota: quando uma classe tem atributos cujo tipo de dados é outra classe do modelo, há uma associação entre elas. Capte essa relação no modelo UML.

E7.4

Crie um modelo para resolver um problema de POO (recorrendo a um diagrama de classes).

Procure utilizar principalmente as capacidades de modelação visual, evitando a edição direta do código (evite o *copy&paste* da solução que já possa ter implementado).

Nota: pretende-se identificar e representar com a UML os elementos da solução; pode implementar a solução (ou não).

Exercício de POO: assunto do exercício 5.1, continuado no 7.2

Quadro 2

```
public class Client {
	private String code;
	private String name;

	public Client(String name, String code) {
		super();
		this.setName(name);
		this.setCode(code);
	}
	public String getCode() {
		return code;
	}
	public String getName() {
		return name;
	}
	public void setCode(String code) {
		this.code = code;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

Quadro 3

```
public class ClientsPortfolio {
	private List<Client> myClientsList;

	public ClientsPortfolio() {
		myClientsList = new ArrayList<>();
	}

	public void addClient(Client newClient) {
		this.myClientsList.add(newClient);
	}
	public int countClients() {
		return this.myClientsList.size();
	}
}
```

Quadro 4

```
public class ClientsDemo {
	public static void main(String[] args) {
		ClientsPortfolio portfolio = new ClientsPortfolio();

		Client client1 = new Client("C101", "Logistica Tartaruga");
		portfolio.addClient(client1);

		Client client2 = new Client("C104", "Jose, Maria & Jesus Lda");
		portfolio.addClient(client2);

		System.out.println("Clients count: "+portfolio.countClients());
	}
}
```