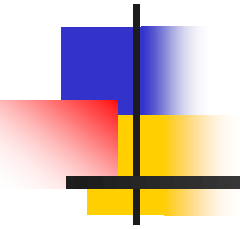


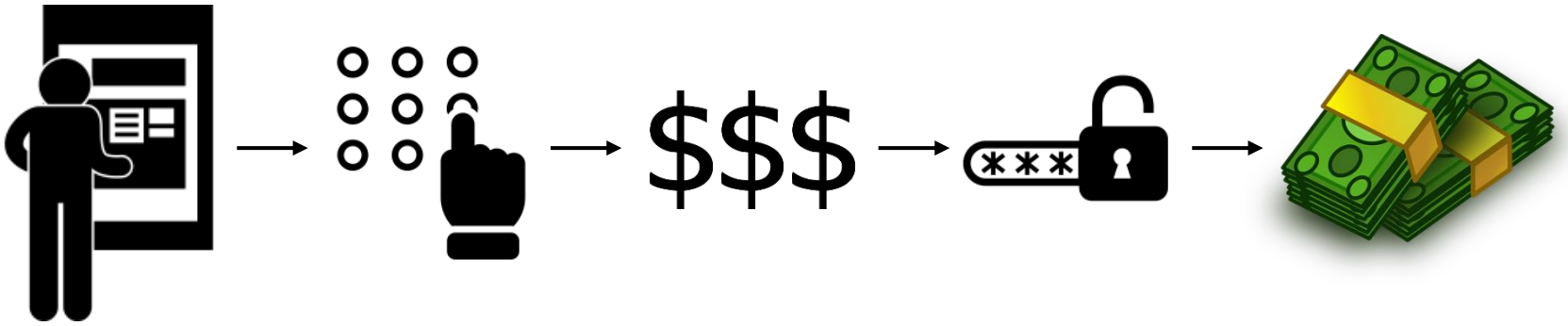
Encapsulamento

Profª. Rachel Reis

rachel@inf.ufpr.br

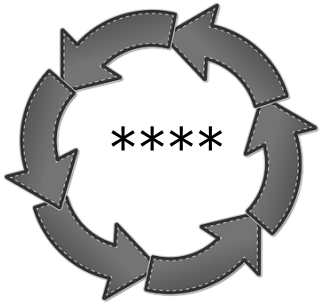
Situação 1

- Sacar dinheiro no caixa eletrônico
  1) Inserir o cartão no caixa eletrônico
  2) Selecionar a opção saque
  3) Digitar o valor
  4) Fornecer a senha
  5) Retirar o dinheiro



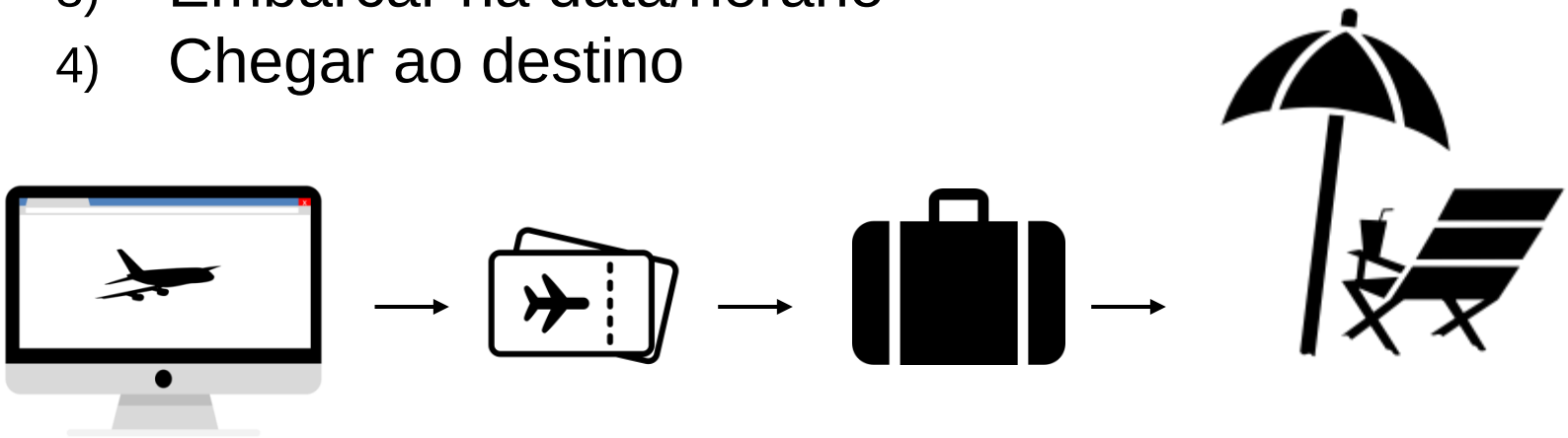
Situação 1

- Não temos que nos preocupar
  - Se o caixa eletrônico possui dinheiro suficiente
  - Se o dinheiro foi debitado da conta corretamente
  - Se o processo de verificação de senha é seguro



Situação 2

- Viagem para o Nordeste
  1) Comprar a passagem
  2) Imprimir o ticket de viagem
  3) Embarcar na data/horário
  4) Chegar ao destino

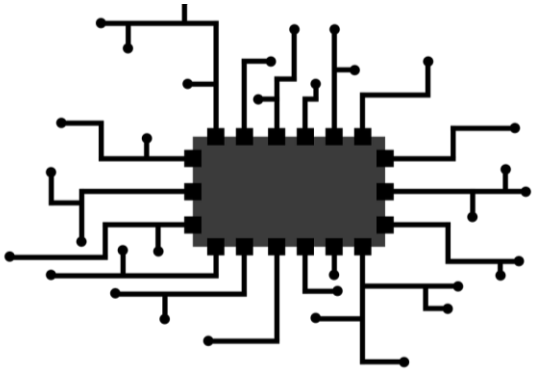


Situação 2

- Não temos que nos preocupar
  - Quais são os acentos disponíveis
  - Qual a melhor rota para chegar ao destino
  - Horário dos avisos de segurança e refeição

Situações 1 e 2

- Descrevem o que é visível ao usuário, ou seja, o que o usuário **tem** que fazer e não **como** o sistema irá realizar o processamento.

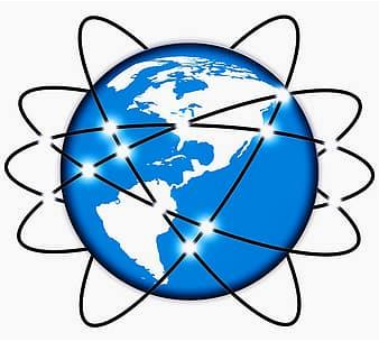


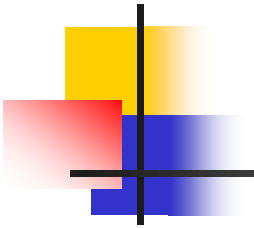
Situações 1 e 2

- Usuário não interfere na dinâmica interna do sistema
  - Exemplo 1:
    - Usuário avisar o banco quando o dinheiro na sua conta estiver acabando.

Situações 1 e 2

- Usuário não interfere na dinâmica interna do sistema
  - Exemplo 2:
    - Usuário verificar qual rota de viagem tem menos chance de ocorrer tempestades.





Encapsulamento

- Definição da Orientação a Objeto que diz que não é preciso conhecer todas a partes de uma classe para entender o seu funcionamento

- Mecanismo que permite separar o funcionamento de sua interface.

- Exemplo: liquidificador

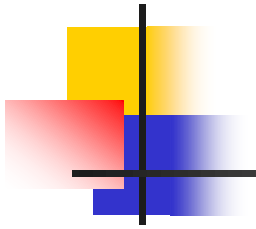
Por que encapsular?

- Questões de segurança
  - Dado sensível.

```
public class Carro
{
    // 'F': flex, 'D': diesel
    char tipoComb;
}
```

Qual é o problema?

```
public class PrincipalCarro{
   public static void main(String[] args)
   {
      Carro x = new Carro();
      x.tipoComb = 'M'; // Valor incorreto
   }
}
```



Por que encapsular?

- Relação de confiança entre as classe
  - Utilizar um método sem conhecer os detalhes de implementação

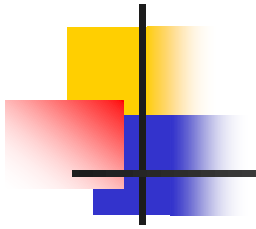
E agora?

```
public class Carro{
    char tipoC;
    ...
    public void setTipoComb(char tipoComb){...}
}
```

```
public class PrincipalCarro
{
     public static void main(String[] args)
     {
       Carro x = new Carro();
       x.setTipoComb('M'); /* Valor incorreto não é
                              atribuído */
     }
}
```

Regra do Encapsulamento

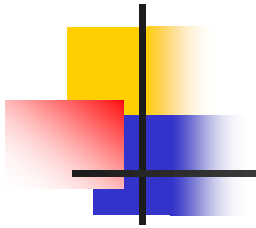
Nenhum objeto pode acessar os campos (atributos) de outro objeto diretamente.



Como encapsular?

- Passo 1
  - **Declarar os atributos da classe como privados**

```
public class Carro
{
    // 'F': flex, 'D': diesel
    private char tipoComb;
}
```



Como encapsular?

- Passo 2
  - **Criar métodos get/set para acessar cada atributo da classe**

Método get - Exemplo

- Objetivo: retornar o valor de um campo encapsulado.

```
public char getTipoComb()
{
    return this.tipoComb;
}
```

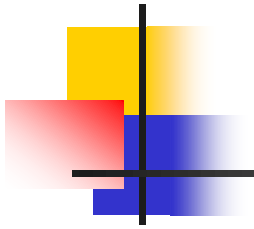
  - Modificador de acesso: public
  - Tipo de retorno: mesmo tipo do campo encapsulado
  - Nome: get + nome do campo encapsulado
  - Parâmetros: não possui.

Método set - Exemplo

- Objetivo: modificar o valor de um campo encapsulado.

```
public void setTipoComb(char tipoComb)
{
   if(tipoComb ==‘F’ || tipoComb ==‘D’)
      this.tipoComb = tipoComb;
}
```

  - Modificador de acesso: public
  - Tipo de retorno: void
  - Nome do método: set + nome do campo encapsulado
  - Parâmetro: valor a ser atribuído para o campo encapsulado.

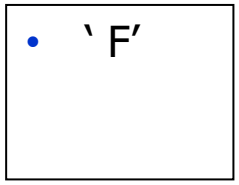


Classe

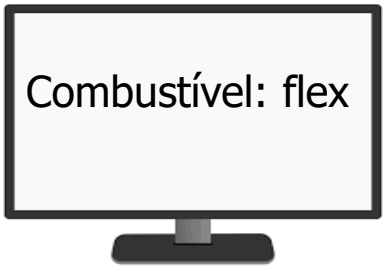
```
public class <Nome da classe>
{
    // Atributos
    // Construtores
    // Métodos get/set
    // Outros métodos
}
```

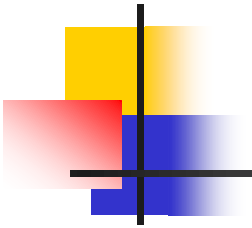
```
public class Carro{

    // Atributos
    private char tipoComb;

    // Método get
    public char getTipoComb()
    {
        return this.tipoComb;
    }

    // Método set
    public void setTipoComb(char tipoComb)
    {
        if(tipoComb ==‘F’ || tipoComb ==‘D’)
            this.tipoComb = tipoComb;
    }
}
```

```
public class PrincipalCarro{
   public static void main(String[] args){
      Carro objeto1 = new Carro();
      objeto1.setTipoComb('F');

      if(objeto1.getTipoComb() == 'F')
          System.out.println("Combustível: flex");
      else if(objeto1.getTipoComb() == 'D')
          System.out.println("Combustível: diesel");
      else
          System.out.println("Não especificado");
   }
}
```



→ **objeto1**





Para praticar...

- Na classe Carro
  - Adicione os atributos: modelo, número de porta, preço
  - Crie métodos get e set para cada atributo
- Na classe PrincipalCarro
  - Crie 3 objetos e armazene em um vetor e peça ao usuário para inserir os dados de cada objeto. Utilize os métodos set.
  - Em seguida, imprima os dados de cada objeto. Utilize os métodos get.

Tipo boolean

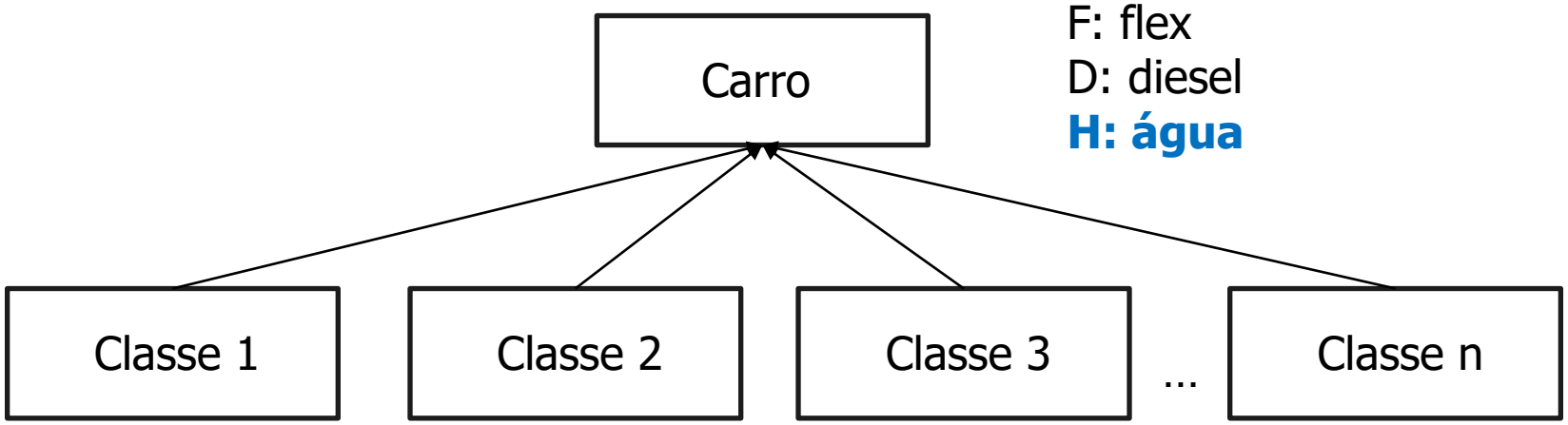
- O padrão dos métodos *get* e *set* **não** vale para as variáveis de tipo *boolean*. Esses atributos são acessados via *is* e *set*.
- Por exemplo: verificar se um carro está ligado

```
public boolean isLigado(){
    return this.ligado;
}
```

```
public void setLigado(boolean ligado){
    this.ligado = ligado;
}
```

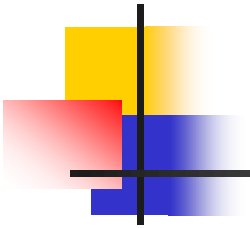
Validação dos valores

- Por que não deixar para validar os dados no método main()???



Papel do Encapsulamento

- Ajudar o programador a dar manutenção no código com o mínimo de esforço.



Referências

- Deitel, P. J.; Deitel, H. M. (2017). Java como programar. 10a edição. São Paulo: Pearson Prentice Hall.

- Barnes, D. J. (2009). Programação orientada a objetos com Java: uma introdução prática usando o BlueJ (4. ed.). São Paulo, SP: Prentice Hall.

- Boratti, I. C. (2007). Programação orientada a objetos em Java. Florianópolis, SC: Visual Books.