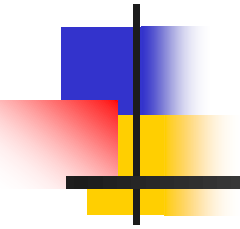


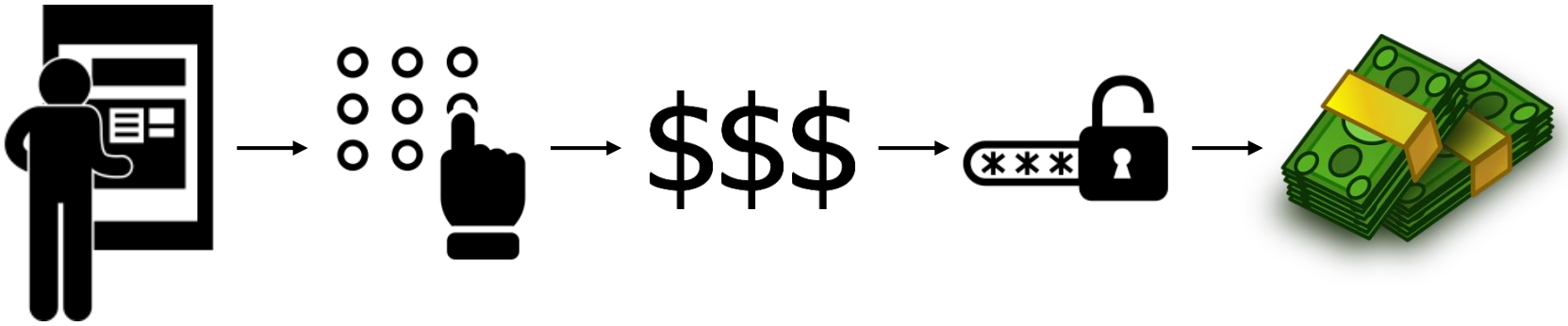
# Encapsulamento

Profª. Rachel Reis

rachel@inf.ufpr.br

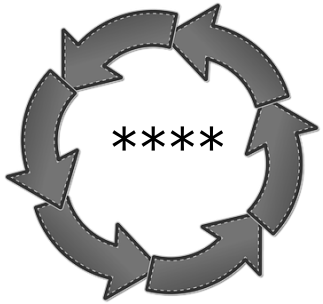
# Situação 1

- 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



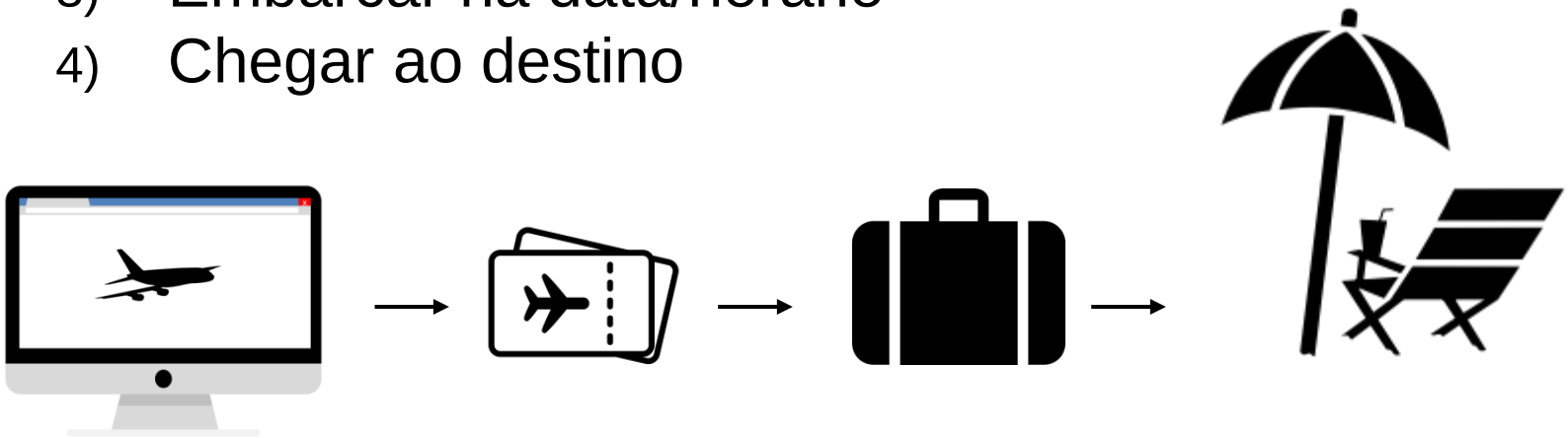
# Situação 1

- 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



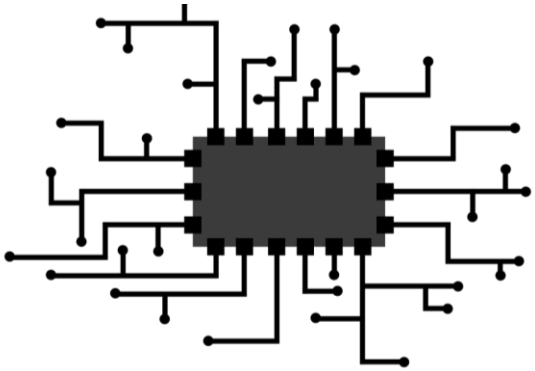
# Situação 2

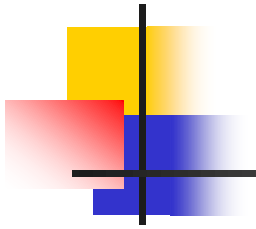
- Não temos que nos preocupar
  - Quais são os acentos disponíveis.
  - Qual a melhor rota para chegar ao destino.
  - Horário dos avisos de segurança e refeição.



# Situações 1 e 2

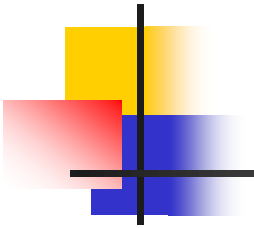
- Descrevem o que é visível ao usuário, ou seja, o que o usuário **tem** que fazer e não **como** o sistema irá realizar o processamento.





# 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.
  - 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?

1) Questões de segurança
   - Dado sensível.

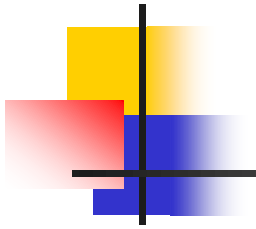
```
public class Carro
{
    // 'F': flex, 'D': diesel
    char tipoComb;
}
```

Qual é o problema?

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

→ objeto1

- 'M'



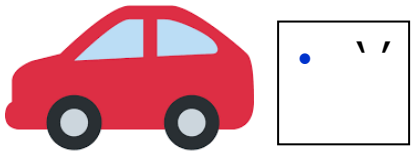
# Por que encapsular?

2) Relação de confiança entre as classes
   - Utilizar um método sem conhecer os detalhes de implementação

E agora?

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

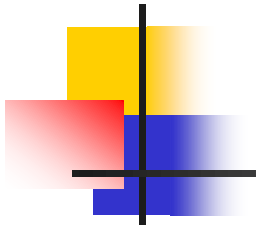
```
public class PrincipalCarro
{
     public static void main(String[] args)
     {
       Carro objeto1 = new Carro();
       objeto1.setTipoComb('M'); // Valor não atribuído
     }
}
```




→ objeto1

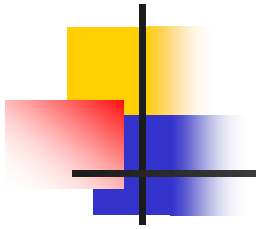
# Regra do Encapsulamento

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



# Modificadores de acesso

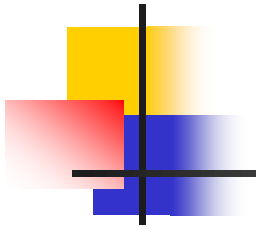
- Tipos:
  - public (+)
  - private (-)
  - protected (#)



# 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;
}
```

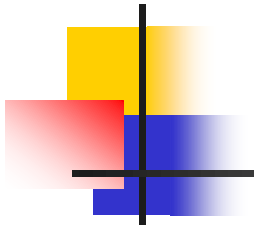
  - Modificador de acesso: public
  - Tipo de retorno: mesmo tipo do campo encapsulado
  - Nome: get + nome do campo encapsulado
  - Parâmetros: não possui.

# 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.



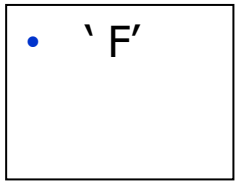
# Classe

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

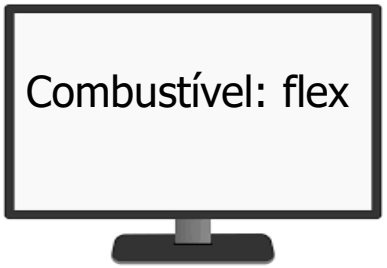
```
public class Carro{

    // 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;
    }
}
```

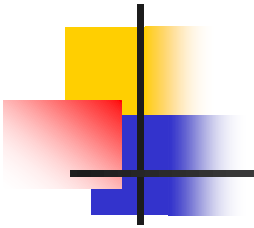
| Carro |
| --- |
| - tipoComb: char |
| + getTipoComb(): char<br>+ setTipoComb(tipoComb:char): void |

```
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 em sala

- Na classe Data (exercício 3, aula prática semana 2)
  - Aplique o conceito de encapsulamento:
    - Declare os atributos dia, mês e ano como privados
    - Crie métodos get para cada atributo
    - Crie métodos set para cada atributo
- Crie 2 objetos e inicialize os campos

```
public class Data{
    // Atributos
    private int dia;
    private int mes;
    private int ano;

    // Métodos get
    public int getDia(){
        return this.dia;
    }
    public int getMes(){
        return this.mes;
    }
    public int getAno(){
        return this.ano;
    }

...
```

Dá para melhorar?

```
public class Data{
    ...
    // Métodos set
    public void setDia(int dia){
      if(dia >=1 && dia <= 31)
          this.dia = dia;
    }

    public void setMes(int mes){
      if(mes >=1 && mes <= 12)
          this.mes = mes;
    }

    public void setAno(int ano){
       if(ano > 0)
          this.ano = ano;
    }
}
```

```
public class Principal{
    public static void main(String[] args){
        // Objeto 1 - data aniversário João
        Data niverJoao = new Data();
        niverJoao.setDia(5);
        niverJoao.setMes(4);
        niverJoao.setAno(2004);

        // Objeto 2 - data domingo de páscoa
        Data domingoPascoa = new Data();
        domingoPascoa.setDia(20);
        domingoPascoa.setMes(4);
        domingoPascoa.setAno(2025);
    }
}
```

# Tipo boolean

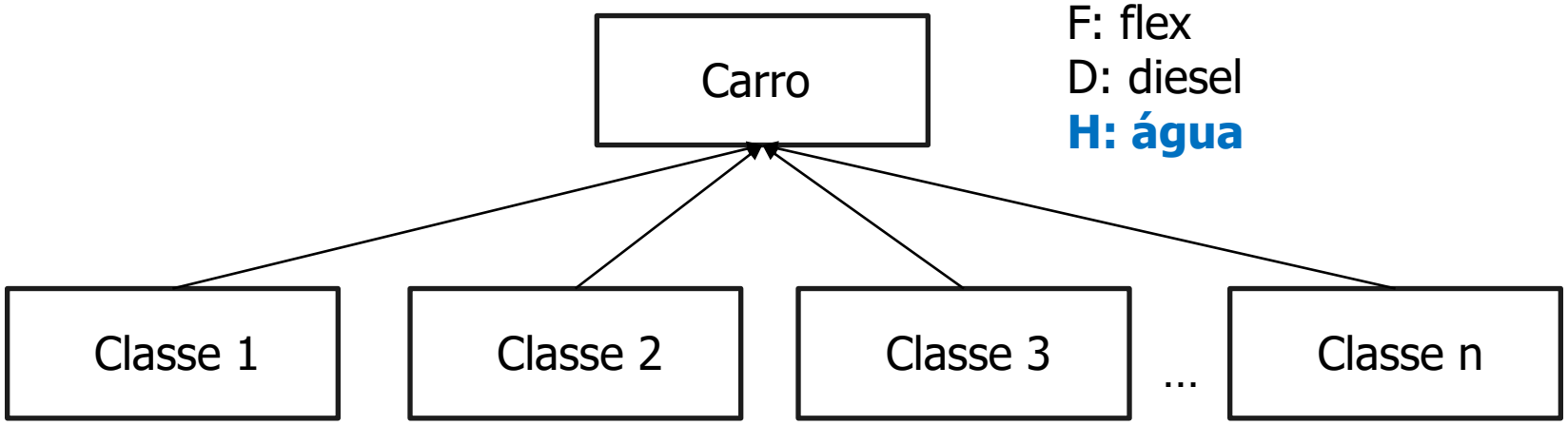
- O padrão dos métodos *get* e *set* **não** vale para as variáveis de tipo *boolean*. Esses atributos são acessados via *is* e *set*.
- Por exemplo: verificar se um carro está ligado

```
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()???

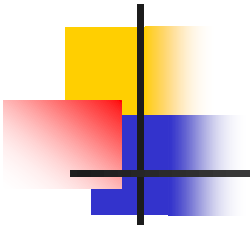


# Papel do Encapsulamento

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

- Implementar os métodos get/set (exercício)

```
public class Funcionario{
    //Atributos
    private String nome;
    private int ano;
    private double salario;

    // Criar os métodos get/set


    // Outros Métodos
    public void cadastrar(String nome, int ano, double salario) {
          this.nome = nome;
          this.ano = ano;
          this.salario = salario;
    }
}
```



# Relembrando: método toString()

- Retorna a representação em string de um objeto.
- Está presente em todas as classes.
- Na forma original, retorna a identidade de um objeto.
- Formato:

nomeclasse@codigohexadecimal

# Relembrando: método toString()

```
Funcionario objeto1 = new Funcionario();
objeto1.setNome("Joao");
objeto1.setAno(2020);
objeto1.setSalario(2550.00);

System.out.println(objeto1);
                    ou
System.out.println(objeto1.toString());
```

- Saída:

```
Funcionario@53d8d10a
```

- Alterando o método toString()

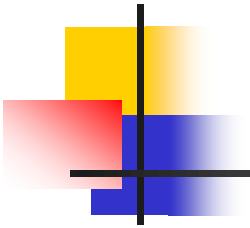
```
public class Funcionario{
      private String nome;
      private int ano;
      private double salario;
      . . .
      public void cadastrar(String nome, int ano, double salario) {
             this.nome = nome;
            this.ano = ano;
            this.salario = salario;
       }

      // Alterando o método toString()
      public String toString()
      {
         return String.format("Nome: %s \nAno: %d \nSalario: %f", this.nome, this.ano,
                                                                    this.salario);
      }
}
```

# Método toString() modificado

```
Funcionario objeto1 = new Funcionario();
objeto1.setNome("Joao");
objeto1.setAno(2020);
objeto1.setSalario(2550.00);

System.out.println(objeto1);
                    ou
System.out.println(objeto1.toString());
```

- Saída:

```
Nome: Joao
Ano: 2020
Salario: 2550.00
```



# 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.