Conceitos de Linguagens de Programação

Orientação a objetos

Um pouco de história...

- Paradigmas de programação
  1. Programação Funcional
  2. Programação Orientada a Objetos
  3. Programação Imperativa

Programação Imperativa

- Rotinas e subrotinas
- A ideia é quebrar o problema em problemas menores e resolver em pequenas *procedures*

Programação Imperativa

- Rotinas e subrotinas
- A ideia é quebrar o problema em problemas menores e resolver em pequenas *procedures*
- **Baixa manutenibilidade**

Programação Imperativa

Considere o esqueleto de programa em C:

```
void fun1(float  r) {
  int s, t;
  ...      <————————————1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <————————————2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <————————————3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```

main chama fun1
fun1 chama fun2
fun2 chama fun3

Programação Imperativa

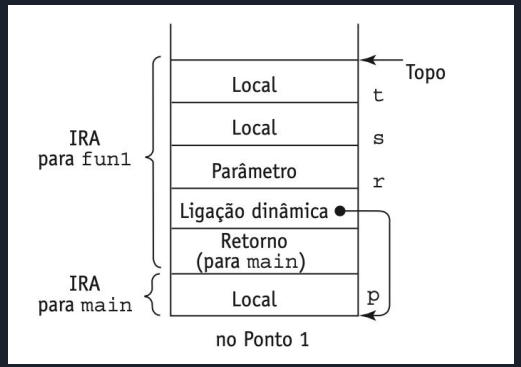
Considere o esqueleto de programa em C:

```
void fun1(float  r) {
  int s, t;
  ...      <--------------1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <--------------2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <--------------3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```



Programação Imperativa

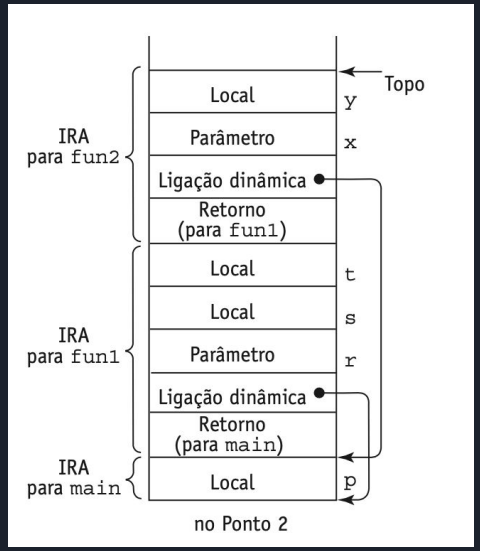
Considere o esqueleto de programa em C:

```
void fun1(float  r) {
  int s, t;
  ...      <-------------1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <-------------2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <-------------3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```



Programação Imperativa

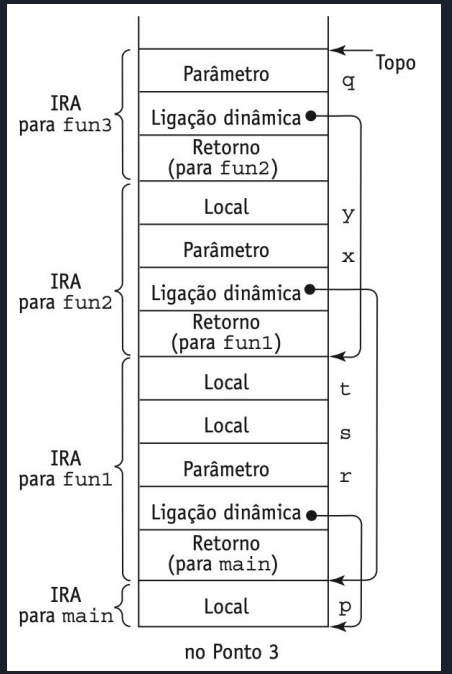
Considere o esqueleto de programa em C:

```
void fun1(float  r) {
  int s, t;
  ...      <-------------1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <-------------2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <-------------3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```



Alta dependência de código

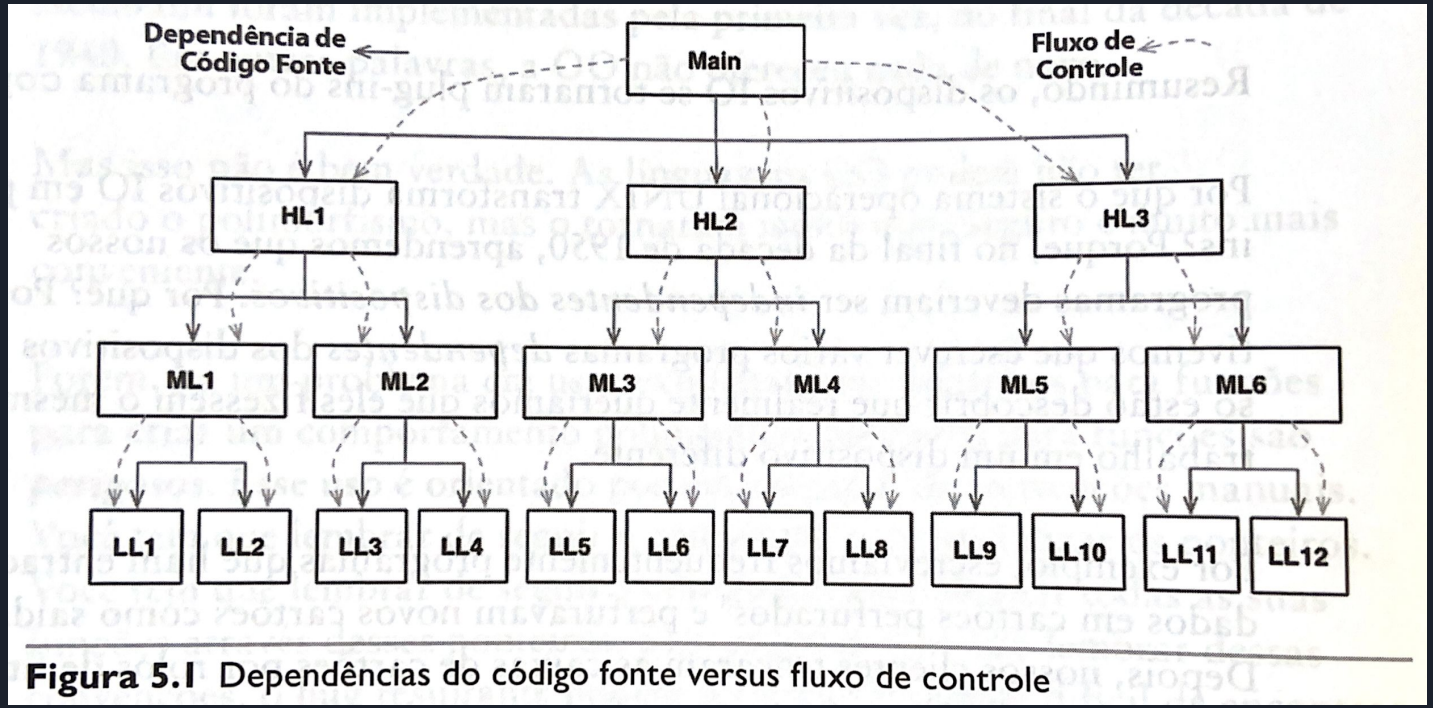


Figura 5.1 Dependências do código fonte versus fluxo de controle

Como vencemos isso?

- Abstração
  - É uma arma contra a complexidade da programação
  - Permite focar nos atributos essenciais, enquanto ignora os subordinados
  - Seu propósito é simplificar o processo de programação

Como vencemos isso?

- Abstração
  - É uma arma contra a complexidade da programação
  - Permite focar nos atributos essenciais, enquanto ignora os subordinados
  - Seu propósito é simplificar o processo de programação

- **Mas o paradigma imperativo não suporta abstração?**

Abstração de processo

- Todos os subprogramas são abstrações de processo
- O programa chamador não conhece os detalhes de um procedimento

```
sortList(list, listLen)
```

Abstração de processo

- Todos os subprogramas são abstrações de processo
- O programa chamador não conhece os detalhes de um procedimento

```
sortList(list, listLen)
```

- O que muda?

Abstração de dados

- Um tipo de dados abstrato é um invólucro que inclui apenas a representação de dados de um tipo de dados específico e os subprogramas que fornecem as operações para esse tipo.
- Unidades de programa que usam tipo de dados abstrato podem declarar variáveis de tal tipo, mesmo que a representação real seja ocultada.

Tipos de Dados Abstratos

- **O usuário não pode manipular diretamente as partes da representação real dos objetos porque essa representação é oculta**
- **A representação dos objetos do tipo é ocultada das unidades de programa que o usam, então as únicas operações diretas possíveis nesses objetos são aquelas fornecidas na definição do tipo**
- Objetos podem ser modificados apenas por meio das operações fornecidas

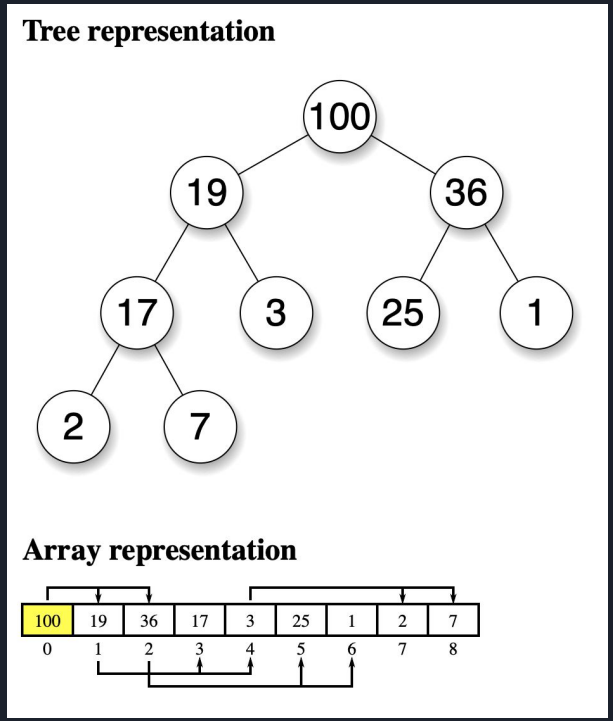
Programação Orientada a Objetos

- Tipos de dados abstratos
- Herança
- Vinculação dinâmica de chamadas a métodos

Programação Orientada a Objetos

- Possibilitou que determinada variável existisse muito depois que a função retornasse
- Heap - coleção de células de armazenamento cuja organização é altamente desorganizada

Programação Orientada a Objetos



Smalltalk - Uma das primeiras linguagens com suporte a OO

- Toda a computação em Smalltalk é feita enviando uma mensagem a um objeto para invocar um de seus métodos.
- Uma resposta para uma mensagem é um objeto, o qual retorna a informação requisitada ou simplesmente notifica o chamador que o processamento solicitado foi completado.

Mensagem?

1. O objeto para o qual a mensagem será enviada;

2. O nome do método que está sendo requisitado;

3. Os parâmetros que devem ser utilizados pelo método.

Conceitos - Vinculação

- É uma associação, como entre um atributo e uma entidade ou entre uma operação e um símbolo
  - Vinculação de nome
  - Vinculação de atributos a variáveis
  - Vinculação de tipos

Método

1. Assinatura
   - Modificadores de acesso
   - Tipo do retorno
   - Nome do método
   - Parâmetros do método
2. Corpo

```
public League getLeague(Integer id) {

    return this.leagues.get(id);

}
```

Conceitos - Escopo e tempo de vida

- Escopo - Faixa de sentenças na qual ela é visível
- Uma variável é visível em uma sentença se ela pode ser referenciada nessa sentença

- Tempo de vida - é o tempo durante o qual ela está vinculada a uma posição de memória

Conceitos - Escopo e tempo de vida

- Algumas vezes, o escopo e o tempo de vida de uma variável parecem ser relacionados

```
public void foo(String input) {
    Integer a = new Integer();
    a = Integer.parseInt(input);
    System.out.println(a);
}
```

Encapsulamento

- Encapsulamento são como “contêineres sintáticos” separados para recursos de software relacionados logicamente
- Esconder detalhes de implementação
- Definir o que pode ser feito

Encapsulamento - Exemplo

```java
public class League {
    2 usages
    private String name;
    2 usages
    private Date beginDate;
    2 usages
    private Date endDate;
    2 usages
    private List<Team> contenders = new ArrayList<>();

    public String getName() { return name; }

    1 usage
    public void setName(String name) { this.name = name; }

    public Date getBeginDate() { return beginDate; }

    1 usage
    public void setBeginDate(Date beginDate) { this.beginDate = beginDate; }

    public Date getEndDate() { return endDate; }

    1 usage
    public void setEndDate(Date endDate) { this.endDate = endDate; }

    1 usage
    public List<Team> getContenders() { return contenders; }

    2 usages
    public void addContender(Team team) { contenders.add(team); }
}
```

Herança

1. Interfaces
    - Define apenas a **especificação** de uma classe
2. Classes abstratas
    - Comportamento semelhante ao da interface
    - Atributos são public static final
    - Métodos sempre públicos
    - Pode conter implementações
3. Classes
    - Onde de fato estão as implementações

Herança - Exemplo

```java
public interface List<E> extends Collection<E> {

    int size();

    boolean isEmpty();

    boolean contains(Object o);

    Iterator<E> iterator();

    Object[] toArray();

    <T> T[] toArray(T[] a);

    boolean add(E e);

    boolean remove(Object o);

    boolean containsAll(Collection<?> c);

    boolean addAll(Collection<? extends E> c);

    boolean addAll(int index, Collection<? extends E> c);

    boolean removeAll(Collection<?> c);

    boolean retainAll(Collection<?> c);
```

Herança - Exemplo

- ArrayList
- LinkedList

- **ArrayList não é um array!**

O que ocorre é que, internamente, ela usa um array como estrutura para armazenar os dados, porém este atributo está propriamente encapsulado e você não tem como acessá-lo.

Polimorfismo

- Termo da biologia que refere ao princípio de quando um organismo ou espécie pode ter várias formas ou estágios
- Em orientação a objetos, subclasses de uma classe podem definir seus próprios comportamentos e ainda compartilhar alguma funcionalidade da classe
- Outro tipo de polimorfismo é o polimorfismo por sobrecarga de métodos, também chamado de poliformismo estático

Polimorfismo - Exemplo (Polimorfismo por herança)

```
public void printDescription(){
    System.out.println("\nBike is " + "in gear " + this.gear
        + " with a cadence of " + this.cadence +
        " and travelling at a speed of " + this.speed + ". ");
}
```

```
public class MountainBike extends Bicycle {
    private String suspension;

    public MountainBike(
                int startCadence,
                int startSpeed,
                int startGear,
                String suspensionType){
        super(startCadence,
              startSpeed,
              startGear);
        this.setSuspension(suspensionType);
    }

    public String getSuspension(){
      return this.suspension;
    }

    public void setSuspension(String suspensionType) {
        this.suspension = suspensionType;
    }

    public void printDescription() {
        super.printDescription();
        System.out.println("The " + "MountainBike has a" +
            getSuspension() + " suspension.");
    }
}
```

Polimorfismo - Exemplo (Polimorfismo por herança)

```
Bike is in gear 5 with a cadence of 20 and travelling at a speed of 10.
The MountainBike has a Dual suspension.
```

Polimorfismo - Exemplo (Polimorfismo de métodos)

```
53      public Team addTeamOnLeague(Integer id, League league) {
54          Team team = this.teams.get(id);
55          team.addTeamOnLeague(league);
56          return team;
57      }
58
        2 usages
59      public Team addTeamOnLeague(Integer id, League league, Integer points) {
60          Team team = this.teams.get(id);
61          team.addTeamOnLeague(league, points);
62          return team;
63      }
```

Java

- Em Java, apenas valores dos tipos primitivos escalares (booleano, caracteres e tipos numéricos) não são objetos.
- Todas as classes Java devem ser subclasses da classe raíz, *Object*, ou de alguma descendente de *Object*.
- Uma **interface** define a especificação de uma classe.
- Interfaces podem ser tratados como tipo, e.g. um método pode especificar um parâmetro que é uma interface.

Tratamento de exceções

- Definimos uma **exceção** como qualquer evento, não usual, errôneo ou não, detectável por hardware ou por software que possa requerer um processamento especial, chamado de **tratamento de exceção**
- **Propagação de exceções** - permite que a exceção levantada em uma unidade do programa ser tratada em outra unidade em seu ancestral dinâmico ou estático