Aula 5

Geomar A. Schreiner
geomarschreiner@gmail.com

Roteiro

- Revisão
  - Polimorfismo
  - Classes Abstratas
- Interface
- Final
- Static
- Classe List
- Exceções

Roteiro

- Revisão
  - Polimorfismo
  - Classes Abstratas
- Interface
- Final
- Static
- **Classe List**
- Exceções

Java

- Arrays

```
nomes[0] = "Arthur";
nomes[1] = "Ford";

int tamanho = nomes.length;

String ultimo = nomes[nomes.length-1];
```

  - Iterar

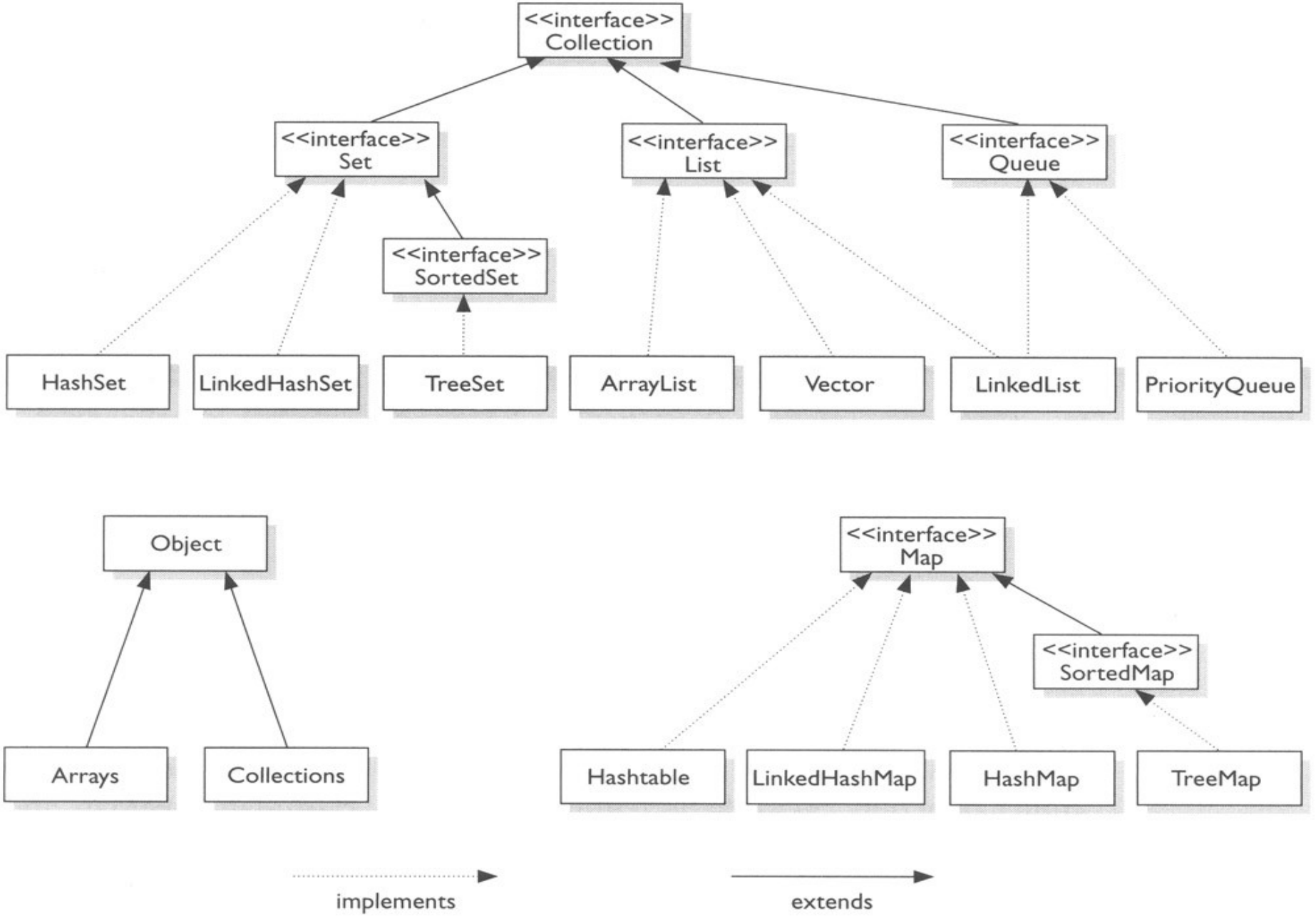
```
String[] array = ...;

for (int i = 0; i<array.length; i++)
{
    String str = array[i];
    //Usa str
}
```

```
for (String str : array)
{
    //Usa str
}
```

Java

- Coleções
  - Java possui uma série de classes que implementam tipos de coleções de dados
    - Vectors
    - Collections <E>
      - Lists <E>
      - Set <E>
      - Queue <E>
    - Maps <K,V>
      - SortedMap <K,V>
      - HashMaps <K,V>

Java


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
Arrays
Collections
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

Java

- Collections
  - Representa uma coleção arbitrária de objetos.
  - Principais métodos:

    int size();
    - Número de elementos da coleção

    boolean contains(Object element);
    - Checa se element pertence à coleção

    boolean add(E element);
    - Adiciona element à coleção

    boolean remove(Object element);
    - Remove element da coleção

    void clear();
    - Remove todos os elementos da coleção

    Iterator<E> iterator();
    - Cria um Iterator, para iterar pelos elementos da coleção

Java

- List
  - Coleção indexada de objetos
  - Métodos

    get(int index);
    - Acessa o i-ésimo elemento da lista

    set(int index, E element);
    - Altera o i-ésimo elemento da lista

    void add(int index, E element);
    - Adiciona um elemento na posição i da lista. Se havia elementos após este índice, eles serão movidos

    Object remove(int index);
    - Remove o i-ésimo elemento da lista

    Int indexOf(Object o);
    - Obtém o índice de um elemento

Java

- List
  - ArrayList
    - Elementos são armazenados de forma contígua, em um array.
    - Acesso indexado rápido.
    - Inserções e remoções no meio da lista são lentos.
  - LinkedList
    - Elementos são armazenados na forma de uma lista encadeada.
    - Acesso indexado péssimo. Precisa percorrer toda a lista.
    - Inserções e remoções no meio da lista são rápidos.

Listas Java

```
public void teste()
{
    List<String> trap = new ArrayList<String>();

    trap.add("Didi");
    trap.add("Dedé");
    trap.add("Mussum");
    trap.add("Zacarias");

    for (int i=0; i < trap.size(); i++)
    {
        String str = trap.get(i);
        System.out.println( str );
    }

    trap.remove(3);
    trap.remove("Mussum")

    for (String s : trap)
        System.out.println( s );

    int idx = lista.indexOf("Didi");

    lista.set(idx, "Beto Carrero");
}
```

Java

- List
  - ArrayList
    - Elementos são armazenados de forma contígua, em um array.
    - Acesso indexado rápido.
    - Inserções e remoções no meio da lista são lentos.
  - LinkedList
    - Elementos são armazenados na forma de uma lista encadeada.
    - Acesso indexado péssimo. Precisa percorrer toda a lista.
    - Inserções e remoções no meio da lista são rápidos.

Java

- Set<E>
  - Lista onde os elementos não podem ser duplicados
    - Geralmente usa o metod toHash (HashSet) ou equal

```
public void teste()
{
    Set<String> s = new HashSet<String>();

    s.add("Cachorro");
    s.add("Gato");
    s.add("Galinha");
    s.add("Gato");

    if (s.contains("Galinha")
    {
        s.remove("Galinha");
    }
    else
    {
        s.add("Cavalo");
        s.add("Boi");
    }
}
```

Java

- Map<K,V>
  - Coleção associativa (pares chave valor)
  - Principais métodos

    V put(K key, V value);
    - Adiciona o objeto value, associado com key

    V get(Object key);
    - Acessa o objeto associado com key

    V remove(Object key);
    - Remove o objeto associado com key

    int size();
    - Número de elementos do Map

Java

```java
public void teste()
{
    Map<String, Pato> m = new HashMap<String, Pato>();

    m.put("Huguinho", new Pato(...));
    m.put("Zezinho", new Pato(...));
    m.put("Luizinho", new Pato(...));


    for (Pato p : m.values())
    {
        System.out.println(p);
    }

    for (String s : m.keySet())
    {
        Pato p = m.get(s);
        System.out.println(s + " -> " + p);
    }


    for (Map.entry<String, Pato> e : m.entrySet())
    {
        System.out.println(e.getKey() + " -> " + e.getValue());
    }
}
```

Roteiro

- Revisão
  - Polimorfismo
  - Classes Abstratas
- Interface
- Final
- Static
- Classe List
- **Exceções**

Java

- Exceções
  - Utilizados para relatar comportamentos anormais no sistema.
    - Pelo user
      - Divisão por zero
      - Caracteres inválidos
    - Pelo código
      - Arquivo inexistente
      - NullPointer
  - Mecanismos de tratamento de exceções permitem que mesmo após o erro o sistema continue executado

Java

- Exceções
  - Throwable
    - classe base de todas as exceções.
    - Principais métodos
      - getMessage(): apresenta o erro
      - printStackTrace(): Apresenta toda a pilha executada até chegar no erro
      - ToString: retorna uma string com a descrição da descrição
  - Exception:
    - subclasse de Thworable;
    - Toda exceção herda da classe de Exception

Java

- Exceções

```
46 try{
47  // trecho do código que pode gerar exceção
48 } catch (IOException objeto) {
49   //tratamento da exceção 1
50 } catch (Exception objeto){
51   //tratamento da exceção 2
52 } finally {
53   //trecho que será executado sempre
54 }
```

Java

- Exceções
  - Métodos são capases de lançar exceções

```
void aumentaTemperatura(int x) throws SuperAquecimento
{
    temperatura += x;

    if (temperatura > 5000)
        throw new SuperAquecimento();
}
```

Java

- Pacotes (Packages)
  - Criam escopos para criação de classes
  - Servem para organizar o código

```
package instrumentos;

public class Teclado
{
    void tocar();
}
```

```
package teste;

import instrumentos.Teclado;
...
Teclado t;
t.tocar();
```

- Denotam o diretório (com base na raiz do projeto) em que um código esta localizado
- Sempre são nomeados em minúsculo

Java - Singleton

- É um classe única, apenas pode ser instanciada uma vez;

```
class Classe
{
  private static final Classe instância = new Classe();

  public static Classe getInstância()
  {
    return instância;
  }

  private Classe() { ... }

  //Demais atributos e métodos da classe em sequência
}

//Uso:
Classe c = Classe.getInstância();
c.qualquerCoisa();
```

Java

- Todo objeto possui um classe raiz
  - Toda classe criada em java deriva implicitamente desta classe
  - Assim essa classe possui alguns métodos base que podem ser sobrescritos

    toString()

    equals(Object a)

    finalize()

    hashCode()

Java

- Todo objeto possui um classe raiz
  - Toda classe criada em java deriva implicitamente desta classe
  - Assim essa classe possui alguns métodos base que podem ser sobrescritos

    toString()

    equals(Object a)

    finalize()

    hashCode()

Java

- ToString

```
class Pessoa
{
    private String nome;
    private int idade;

    public String toString()
    {
        return nome + ", com "
                + idade;
    }
}
```

```
Pessoa p = new Pessoa(...);
...
System.out.println(p);
```

```
Joãozinho, com 14
```

Java

- Todo objeto possui um classe raiz
  - Toda classe criada em java deriva implicitamente desta classe
  - Assim essa classe possui alguns métodos base que podem ser sobrescritos

    toString()

    equals(Object a)

    finalize()

    hashCode()

Java

- Equals

```
class Pessoa
{
    private String nome;
    private int idade;

    public boolean equals(Object o)
    {
        if (this == o)
             return true;
        else if (o == null || getClass() != o.getClass())
        return false;
        else
        {
            Pessoa p = (Pessoa) o;
            return nome.equals(p.nome) && idade == p.idade;
        }
    }
}
```

Exercício

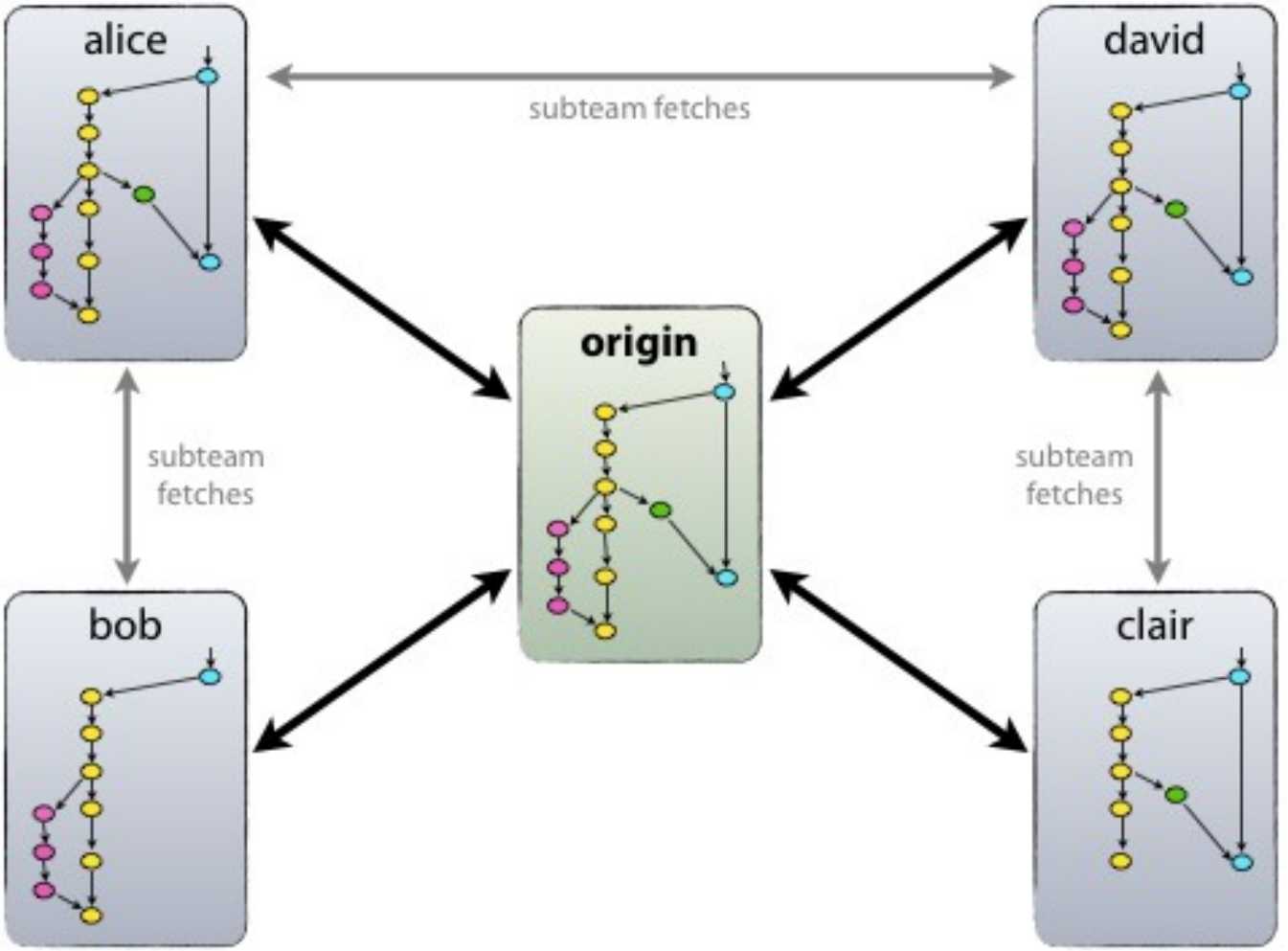
- Você foi encarregado de desenvolver um software de desenho CAD. Claro que você não fará isso sozinha, seus amigos da Nasa vão te ajudar. Porém você foi incumbido de fazer uma versão funcional das classes utilizadas no software final. Durante o levantamento de requisitos, você descobriu o seguinte. O sistema inteiro funciona dentro de um plano cartesiano, ou seja, elementos são posicionados com coordenadas (x, y). A forma mais primitiva de desenho é o ponto, que é usado sozinho ou em conjunto para compor e manipular outras formas mais complexas. Por exemplo, uma linha é composta por dois pontos, o triângulo é feito com 3 pontos, etc. Os usuários informaram que precisam, além da forma primitiva ponto, as seguintes formas complexas: quadrado, retângulo, círculo e triângulo. Algumas das formas podem ser rotacionadas pelo usuário (quadrado e retângulo), outras não; algumas das formas podem ser "aumentadas" ou "encolhidas" (círculo e triângulo). Por fim, você descobriu que o programa pode ter um número infinito de formas na tela, porém todas devem ser guardadas num mesmo vetor na memória do programa. No futuro, novas formas complexas serão adicionadas por outros desenvolvedores.
- Você deve criar a estrutura básica de classes das formas geométricas. Todas as formas devem possuir os seguintes métodos:
  - podeRotaciona: retorna true se a classe pode ser rotacionada ou false caso contrario.
  - podeAumentar: retorna true se a classe pode ser aumentada ou false caso contrário.
  - podeDiminuir: retorna true se a classe pode ser diminuída ou false caso contrário.
- Seu programa deve permitir que o usuário cadastre e exclua formas geométricas. Outra feature necessária é que o usuário possa listar todas as formas que já foram cadastradas (na ordem de cadastro).

Introdução ao Git

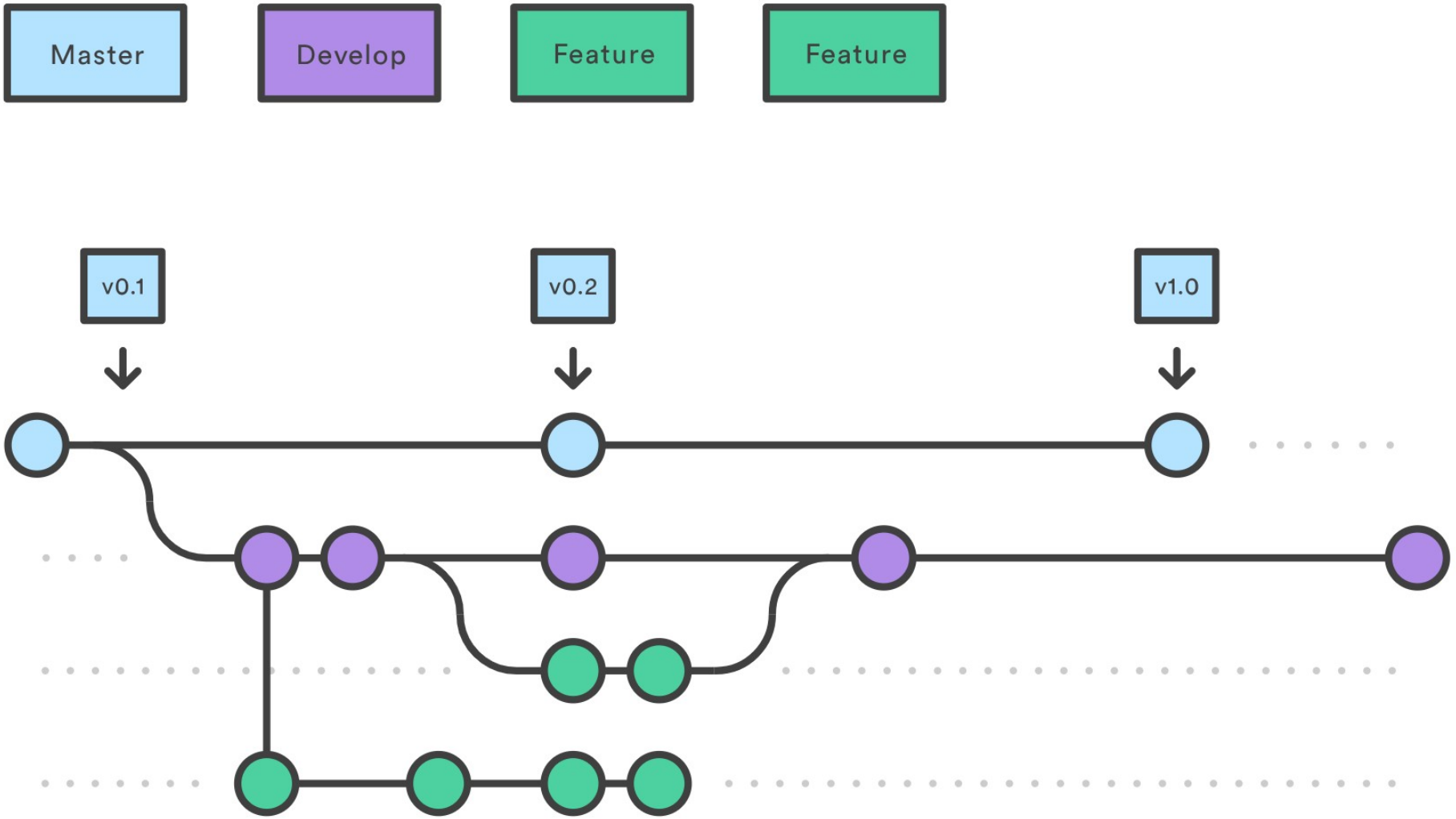
Git

- Git é um sistema de versionamento distribuido
  - Organizar várias versões de documentos
- Criado por Linus Torvalds
- Durante o desenvolvimento permite saber
  - O que mudou de uma versão para outra
  - Quem fez as modificações?
  - Porque mudou?

Git


alice
david
subteam fetches
origin
subteam
fetches
subteam
fetches
bob
clair

Git



Git

- Instalação GIT
  - Ubuntu

    ```
    # apt install git
    ```

  - Windows
    - https://git-scm.com/download/win

  - Criar conta GitHub
    - https://github.com/
    - Tour e Criar o primeiro Projeto

Git

- Comandos Básicos
  - Copiar um repositório para a minha máquina

    git clone LINK
  - Adicionar um documento

    git add arquivo.code
  - Remover um documento

    git rm arquivo.code
  - Verificar as modificações do meu branch

    git status
  - Salvar minhas alterações

    git commit -m ‘lero lero’
    - Opção ‘-a’

Git

- Comandos Básicos
  - Configurar username

    git config --global user.name ‘BiruBiru’

  - Configurar o e-mail

    git config - - global user.email “birubiru@gmail.com”

  - Ver os ultimos commits

    git log

Git

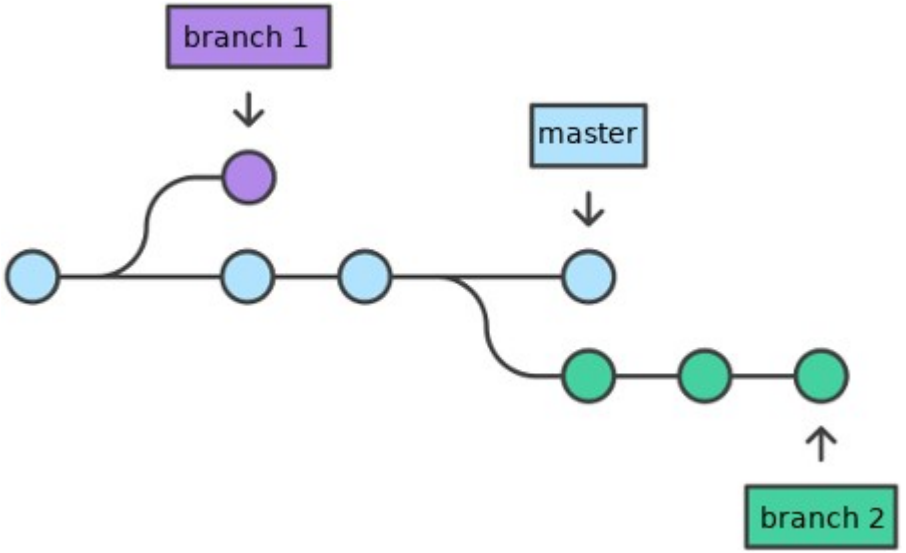
- Comandos Básicos
  - Enviar as informações para o remote (github)

    git push **origin master**
  - Atualizar as informações atuais com o servidor

    git pull origin master

Git

- Branches



- Criar branch

    git branch -a nome

Git

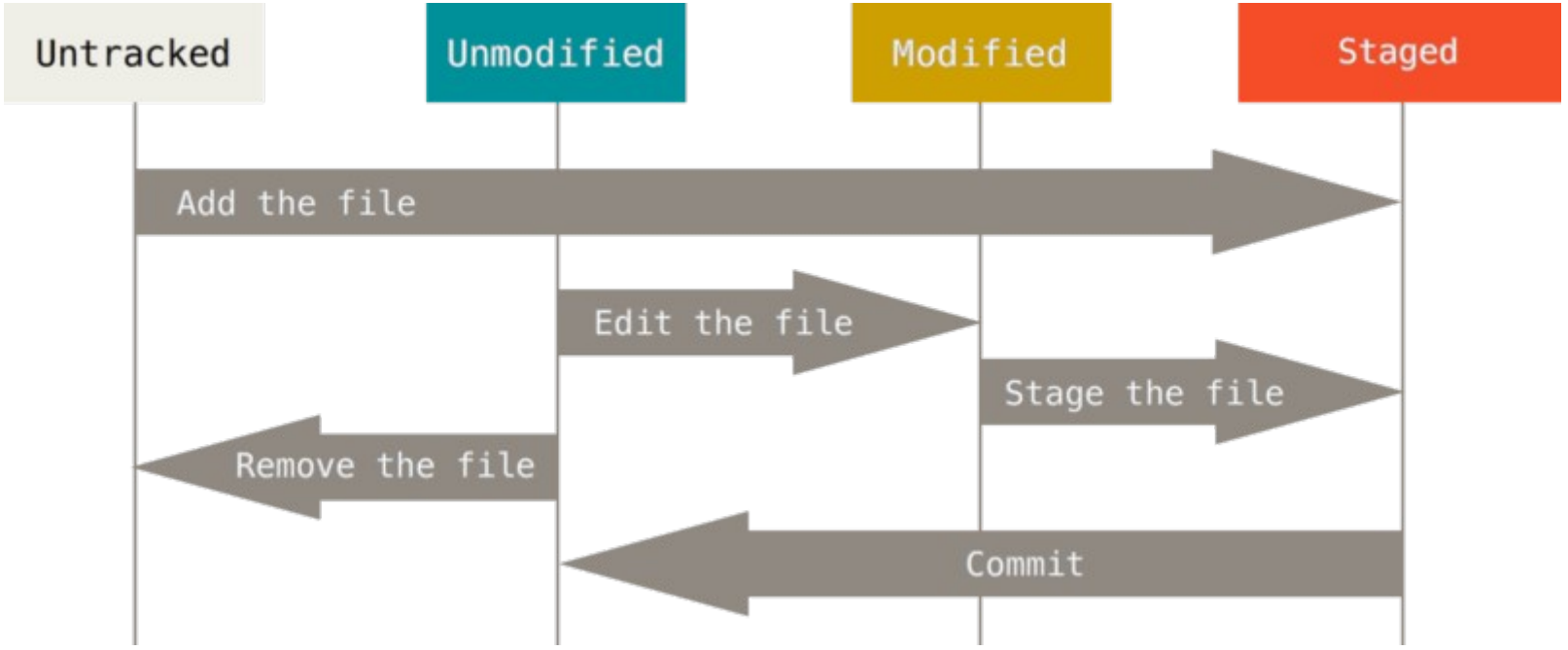
- Comandos Básicos
  - Alternar entre diferente branches

    git checkout nome
  - Voltar para uma versao anterior

    git checkout <commit> <documento>

Git

- Importante
  - Todos os documentos serão visualizados pelo GIT



  - Arquivos de compilação também.

Git

- Importante
  - Para resolver isso foi criado o gitignore
    - Basta criar arquivo .gitignore na raiz do projeto
      - No arquivo devem ser inseridas as extensões que o git deve ignorar
    - Ex

.gitignore Java

```
1  ###################################
2  ## Java
3  ###################################
4  .mtj.tmp/
5  *.class
6  *.jar
7  *.war
8  *.ear
9  *.nar
10 hs_err_pid*
```