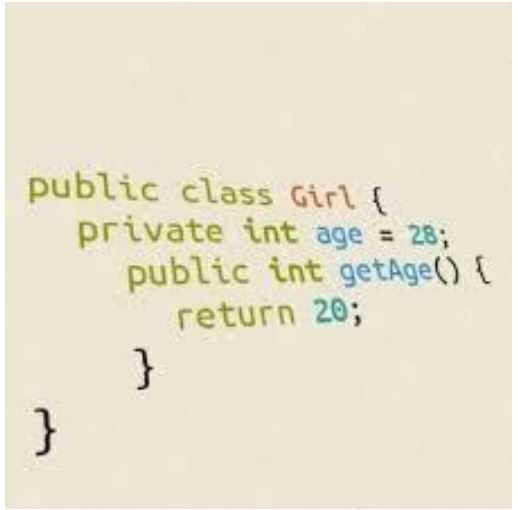
JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Part 8

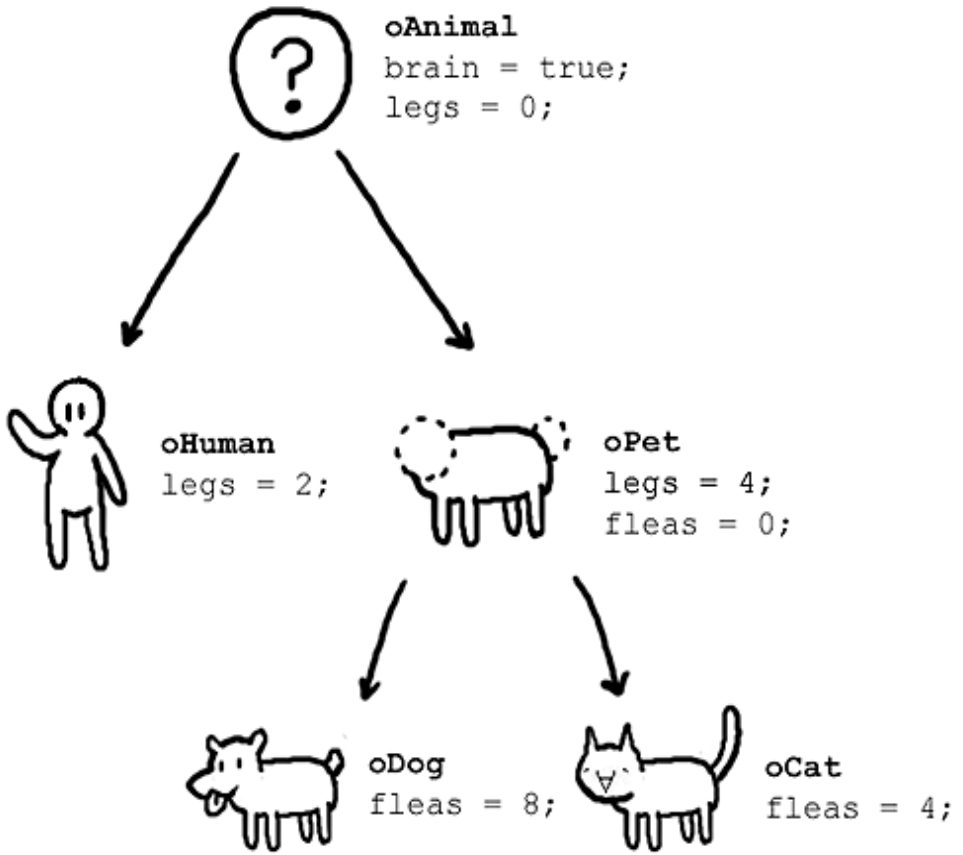
Inheritance

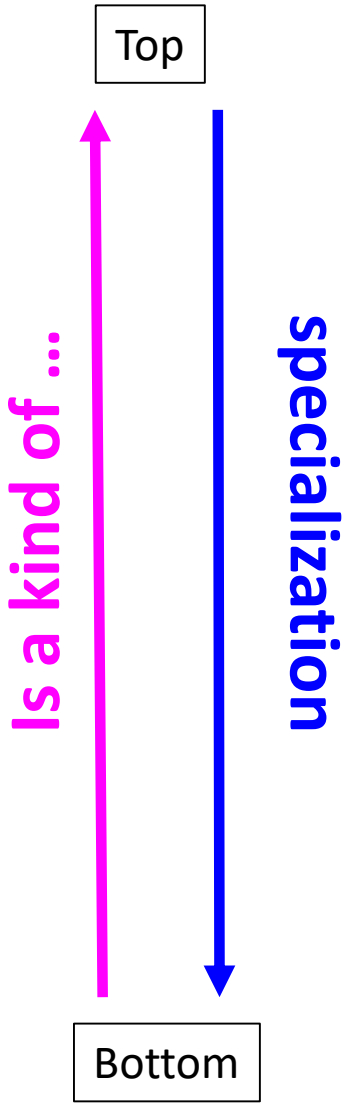


Exam question samples

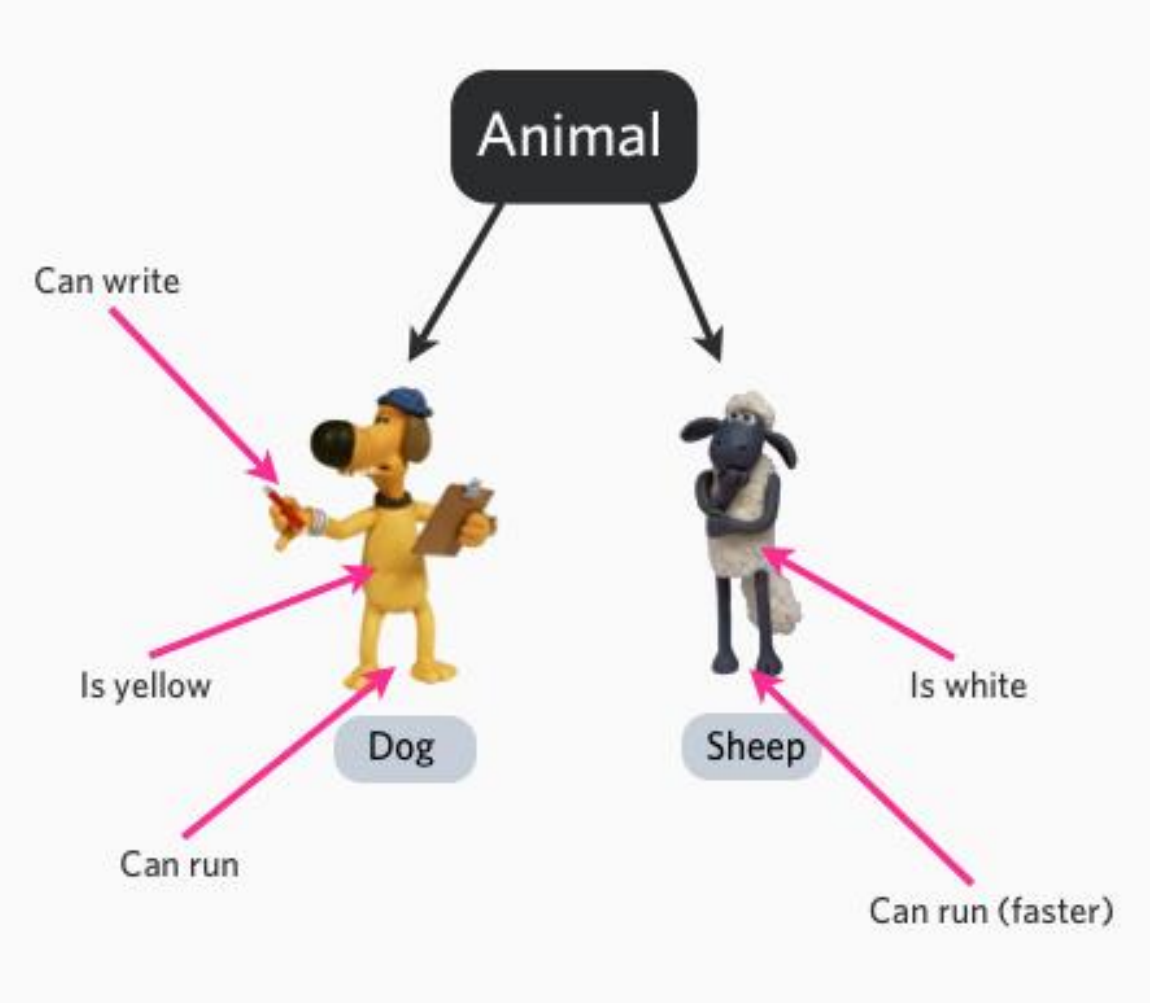
- What is the garbage collector, how does it works?
- Explain why the toString method is expected to be overridden?
- Explain privacy leak in Java, how can it be avoided?
- Explain the *finalize ( )* method.
- What is a package? How is it used?
- What are command line arguments?
- What differences between instance variable and local variable?
- List the steps for creating an object of a class.
- What is a default constructor? When does it exist?
- Give the definition of a static method. What uses can be made of such method? What is not authorized?

Inheritance





Inheritance



Inheritance

- **Inheritance** is the process by which a new class (the **derived class**) is created from another class (the **base class**)
- A derived class (daughter class) gets access to **all** the **non-private** instances variables and to **all** the **non-private** methods of the **base class**
- A derived class could have **additional** instances variables or methods
- Inheritance is often represented through a **class hierarchy** (class diagram)

Inheritance

CowBoy

private String name;
private int popularity;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Robber

private String name;
private boolean isInJail;
private int rewardValue;
private String look;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Public void kidnapLady(Lady) { --- }

Lady

private String name;
private boolean isKidnapped;
private Color dressColor;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

public void changeDress(Color c) { --- }
public void beKidnapped (Robber) { --- }

Inheritance

Object oriented programing : Make it simple

Common attributes can (should) be factorized

CowBoy

private String name;
private int popularity;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Robber

private String name;
private boolean isInJail;
private int rewardValue;
private String look;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Public void kidnapLady(Lady) { --- }

Lady

private String name;
private boolean isKidnapped;
private Color dressColor;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

public void changeDress(Color c) { --- }
public void beKidnapped (Robber) { --- }

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Character

protected String name;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

CowBoy

private String name;
private int popularity;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Robber

private String name;
private boolean isInJail;
private int rewardValue;
private String look;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

Public void kidnapLady(Lady) { --- }

Lady

private String name;
private boolean isKidnapped;
private Color dressColor;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

public void changeDress(Color c) { --- }
public void beKidnapped (Robber) { --- }

Character

protected String name;

public void canTalk ( ) { --- }
public void introduce( ) { --- }

CowBoy

private int popularity;

Robber

private boolean isInJail;
private int rewardValue;
private String look;

Public void kidnapLady(Lady) { --- }

Lady

private boolean isKidnapped;
private Color dressColor;

public void changeDress(Color c) { --- }
public void beKidnapped (Robber) { --- }

Inheritance

- Syntax for defining a derived class:

```
public class DaughterClassName extends MotherClassName {
        Declarations of new class variables;
        Declarations of new instance variables;
        Declarations of new methods ;
        Overriding of inherited methods;
}
```

- In constructors of the derived class, you could use constructors of the base class by using the keyword **super(**Parameters_List**)**

- In Java, you could only inherit from one class and only one

```java
/**
 *
 * @author mikael.morelle
 */
public class Character {

    protected String name;

    /**
     * Character constructor, it is not possible
     * to create a character without a name.
     * @param name the name of the Character
     */
    public Character(String name){
        this.name = name;
        introduceYourself();
    }

    /**
     * Accessor of the name attribute
     * @return the name of the character
     */
    public String getName(){return this.name;}

    /**
     * method that allows the character to talk and
     * to say something on the output screen
     * @param say what the character says
     */
    public void talk(String say){
        System.out.println(name + " : " + say);
    }

    /**
     * Each time a character is created, he/she has
     * to introduce to the already existing population
     */
    public void introduceYourself(){
        talk("Hello Everyone, my name is " + name + ". I am the newest citizen.");
    }

}
```

```java
package western;
/**
 * @author mikael.morelle
 */
public class Western {
    public static void main(String[] args) {

        Character luke = new Character("Lucky Luke");
        Character sam = new Character("Samantha");
        Character butch = new Character("Butch Cassidy");

        System.out.print("\n");
        luke.talk("Nice to meet you "+sam.getName());
        luke.talk("Nice to meet you "+ butch.getName());
    }
}
```

western.Western > main > luke

ut - Western (run)

```
run:
Lucky Luke : Hello Everyone, my name is Lucky Luke. I am the newest citizen.
Samantha : Hello Everyone, my name is Samantha. I am the newest citizen.
Butch Cassidy : Hello Everyone, my name is Butch Cassidy. I am the newest citizen.

Lucky Luke : Nice to meet you Samantha
Lucky Luke : Nice to meet you Butch Cassidy
BUILD SUCCESSFUL (total time: 0 seconds)
```

To create an object of the daughter class, you have to instanciate an object of the mother class first

We redefine (provide another implementation) of the inherited method

```java
public class Cowboy extends Character{

    private int popularity=10;
    private String adjective;
    /** CowBoy constructor, it is not possible ...6 lines */
    public Cowboy(String name, String adjective){
        super(name);
        this.adjective = adjective;
        introduceYourself();
    }

    /** Accessor of the popularity attribute ...4 lines */
    public int getPopularity(){
        return this.popularity;
    }
    /** Accessor of the adjective attribute ...4 lines */
    public String getAdjective(){
        return this.adjective;
    }

    /** Cowboy introduce themselves in a kind way ...3 lines */
    @Override
    public void introduceYourself() {
        super.introduceYourself();
        talk("People used to call me : " + this.name + " the " + this.adjective);
    }

    /** Method returning the string giving the manner of calling a cowboy ...4 lines */
    public String whoAreYou(){
        return this.getName() + " the " + this.getAdjective();
    }
}
```

```java
public class Western {
    public static void main(String[] args) {

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        System.out.println("");
    }
}
```

ut - Western (run)

```
run:
Lucky Luke : Hello Everyone, my name is Lucky Luke. I am the newest citizen.
Lucky Luke : People used to call me : Lucky Luke the Brave

BUILD SUCCESSFUL (total time: 0 seconds)
```

To create an object of the daughter class, you have to instanciate an object of the mother class first

We redefine (provide another implementation) of the inherited method

We redefine (provide another implementation) of the inherited method

```java
public class Robber extends Character{
    private boolean isInJail = false;
    private int rewardValue;
    private int numberOfKidnapperLady;
    private String look;

    public Robber(String name, String look){
        super(name);
        this.look = look;
        this.rewardValue = 10;

        introduceYourself();
    }
    /** robbers introduce themselves in an angry way ...3 lines */
    @Override
    public void introduceYourself() {

        talk("My name is " + this.name + " and I look "+ this.look);
        talk("Don't bother me");
    }

    /** Robbers always end their talk by a '!!!' sign ...4 lines */
    @Override
    public void talk(String say){
        System.out.println(name + " : " + say + "!!!");
    }
    /** In westerns Robbers kidnap ladies  ...4 lines */
    public void kidnapLady(Lady lady){
        this.rewardValue += 10;
        this.numberOfKidnapperLady++;

        lady.hasBeenKidnapped(this);

        talk("You're mine now " + lady.whoAreYou());

    }
    /** Method returning the string giving the manner of calling a robber ...4 lines */
     public String whoAreYou(){
        return this.getName() + " the " + this.look;
    }
}
```

```java
public class Western {
    public static void main(String[] args) {

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        System.out.println("");

        Robber butch = new Robber("Butch Cassidy", "Wicked");
        System.out.println("");
    }
}
```

western.Western > main >

ut - Western (run)

```
run:
Lucky Luke : Hello Everyone, my name is Lucky Luke. I am the newest citizen.
Lucky Luke : People used to call me : Lucky Luke the Brave

Butch Cassidy : My name is Butch Cassidy and I look Wicked!!!
Butch Cassidy : Don't bother me!!!

BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class Lady extends Character{
    private String dressColor;
    private boolean isKidnapped;
    /** Lady constructor, it is not possible ...6 lines */
    public Lady(String name, String dressColor){
        super(name);
        this.isKidnapped=false;
        this.dressColor = dressColor;

        introduceYourself();
    }

     /** Ladies introduce themselves in a very lady way ...3 lines */
    @Override
    public void introduceYourself() {
        super.introduceYourself();
        talk("Look at me, I have just bought this wonderful "+dressColor+" dress. I am so happy.");
    }

    /** Unfortunately in many westerns at some point, ladies are kidnapped by a  robber ...4 lines */
    public void hasBeenKidnapped(Robber robber){
        this.isKidnapped = true;
        talk("Help me, Help me");
        talk(robber.whoAreYou() + " is kidnapping me ... Noooooo....");
    }

    /** Method returning the string giving the manner of calling a lady ...4 lines */
     public String whoAreYou(){
        return "Lady " + this.dressColor;
    }
}
```

```java
public class Western {
    public static void main(String[] args) {

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        System.out.println("");

        Robber butch = new Robber("Butch Cassidy", "Wicked");
        System.out.println("");

        Lady sam = new Lady("Samantha", "Pink");
        System.out.println("");
    }
}
```

western.Western > main >

ut - Western (run)

```
run:
Lucky Luke : Hello Everyone, my name is Lucky Luke. I am the newest citizen.
Lucky Luke : People used to call me : Lucky Luke the Brave

Butch Cassidy : My name is Butch Cassidy and I look Wicked!!!
Butch Cassidy : Don't bother me!!!

Samantha : Hello Everyone, my name is Samantha. I am the newest citizen.
Samantha : Look at me, I have just bought this wonderful Pink dress. I am so happy.

BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class Western {
    public static void main(String[] args) {

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        System.out.println("");

        Robber butch = new Robber("Butch Cassidy", "Wicked");
        System.out.println("");

        Lady sam = new Lady("Samantha", "Pink");
        System.out.println("");

        butch.kidnapLady(sam);
    }
}
```

western.Western › main ›

t - Western (run)

```
run:
Lucky Luke : Hello Everyone, my name is Lucky Luke. I am the newest citizen.
Lucky Luke : People used to call me : Lucky Luke the Brave

Butch Cassidy : My name is Butch Cassidy and I look Wicked!!!
Butch Cassidy : Don't bother me!!!

Samantha : Hello Everyone, my name is Samantha. I am the newest citizen.
Samantha : Look at me, I have just bought this wonderful Pink dress. I am so happy.

Samantha : Help me, Help me
Samantha : Butch Cassidy the Wicked is kidnapping me ... Noooooo....
Butch Cassidy : You're mine now Lady Pink!!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Western stories generator

package places;

package interactions;

package people;

super constructor

- A derived class uses a constructor from the base class to initialize all the inherited data
- In order to invoke a constructor from the base class, it uses a special syntax:

```
public Lady(String name, String dressColor){
    super(name);
    this.isKidnapped=false;
    this.dressColor = dressColor;

    introduceYourself();
}
```

- In the above example, super(name); is a call to the base class constructor (Character)

```
public Character(String name){
    this.name = name;
    introduceYourself();
}
```

super constructor

- A call to the base class constructor can never use the name of the base class, but uses the keyword **super** instead
- A call to **super** must always be the first action taken in a constructor definition
- An instance variable of a daughter class cannot be used as an argument to **super**
- If a derived class constructor does not include an invocation of super, then the no-argument constructor of the base class will automatically be invoked (when it exists)
- This can result in an error if the base class has not defined a no-argument constructor
- Since the inherited instance variables should be initialized, and the base class constructor is designed to do that, then an explicit call to super should always be used

this constructor

- Within a constructor, **this** can be used as a name for invoking another constructor in the same class
- The same restrictions on how to use a call to **super** apply to the **this** constructor

If it is necessary to include a call to both **super** and **this**, the call using this must be made first, and then the constructor that is called must call super as its first action

```
public class CowBoy extends Character{
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

    public CowBoy(String name, String adjective,SoftDrink favouriteDrink ){
        this(name,adjective);
        this.favouriteDrink = favouriteDrink;
    }

    public CowBoy(String name, String adjective){
        super(name);
        this.adjective = adjective;
        this.favouriteDrink = SoftDrink.WATER;
    }
}
```

dynamic polymorphism

- An inherited method can be redefine in a derived class trough **overriding**
- To override a method, you have just to **redefine it** in the derived class (@override)
- The modifier **final**, in the definition of a method, indicates that the method could not be redefined in a derived class
- Change of the access permission of an overriden method is authorized in the derived class
- **Private** methods are **never inherited**
- **Protected** methods are inherited

```java
public class Character {
    protected String name;

    /** Character constructor, it is not possible ...5 lines */
    public Character(String name){
        this.name = name;
    }

    /** Accessor of the name attribute ...4 lines */
    protected String getName(){
        return this.name;
    }

    /** method that allows the character to talk and ...5 lines */
    public void talk(String say){
        System.out.println(name + " : " + say);
    }

    /** Each time a character is created, he/she has ...4 lines */
    public void introduceYourself(){
        talk("Hello Everyone, my name is " + name + ". I am the newest citizen.");
    }
}
```

```java
public class Cowboy extends Character{

    private int popularity=10;
    private String adjective;
    /** CowBoy constructor, it is not possible ...6 lines */
    public Cowboy(String name, String adjective){
        super(name);
        this.adjective = adjective;
        introduceYourself();
    }

    /** Accessor of the popularity attribute ...4 lines */
    public int getPopularity(){
        return this.popularity;
    }
    /** Accessor of the adjective attribute ...4 lines */
    public String getAdjective(){
        return this.adjective;
    }

    /** Cowboy introduce themselves in a kind way ...3 lines */
    @Override
    public void introduceYourself() {
        super.introduceYourself();
        talk("People used to call me : " + this.name + " the " + this.adjective);
    }

    /** Method returning the string giving the manner of calling a cowboy ...4 lines */
    public String whoAreYou(){
        return this.getName() + " the " + this.getAdjective();
    }
}
```

introduceYourself ( ) : partially redefined.
It is a completed version

talk ( ) : not redefined, Character's version is used.

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

```java
public class Character {
    protected String name;

    /** Character constructor, it is not possible ...5 lines */
    public Character(String name){
        this.name = name;
    }

    /** Accessor of the name attribute ...4 lines */
    protected String getName(){
        return this.name;
    }

    /** method that allows the character to talk and ...5 lines */
    public void talk(String say){
        System.out.println(name + " : " + say);
    }

    /** Each time a character is created, he/she has ...4 lines */
    public void introduceYourself(){
        talk("Hello Everyone, my name is " + name + ". I am the newest citizen.");
    }
}
```

introduceYourself ( ) : fully redefined.
It is a new version

talk ( ) : fully redefined.
It is a new version

```java
public class Robber extends Character{
    private boolean isInJail = false;
    private int rewardValue;
    private int numberOfKidnapperLady;
    private String look;

    public Robber(String name, String look){
        super(name);
        this.look = look;
        this.rewardValue = 10;

        introduceYourself();
    }
    /** robbers introduce themselves in an angry way ...3 lines */
    @Override
    public void introduceYourself() {

        talk("My name is " + this.name + " and I look "+ this.look);
        talk("Don't bother me");
    }

    /** Robbers always end their talk by a '!!!' sign ...4 lines */
    @Override
    public void talk(String say){
        System.out.println(name + " : " + say + "!!!");
    }
    /** In westerns Robbers kidnap ladies  ...4 lines */
    public void kidnapLady(Lady lady){
        this.rewardValue += 10;
        this.numberOfKidnapperLady++;

        lady.hasBeenKidnapped(this);

        talk("You're mine now " + lady.whoAreYou());

    }
    /** Method returning the string giving the manner of calling a robber ...4 lines */
     public String whoAreYou(){
        return this.getName() + " the " + this.look;
    }
}
```

Composition

- Composition is the word used to define that an instance of a class could use other instances of other classes as its variables instances
- In other words, composition is referencing other instances of classes
- A composition is a relationship of type **“has a”** between classes
- Example :
  - Person is a generic class for defining a person
  - A person **“has a”** mother which could be defined as an instance of class Person
  - A person **“has a”** name which could be defined as an instance of class String
  - A person **“has a”** car which could be defined as an instance of generic class Vehicule
- These relationships could be defined as instance variables in the class Person

```java
public class Cowboy extends Character{

    private int popularity=10;
    private String adjective;

    private Gun theGun = new Gun();

    /**
     * CowBoy constructor, it is not possible
     * to create a cowboy without a name and an adjective
     * @param name the name of the Character
     * @param adjective the adjective of the cowboy
     */
    public Cowboy(String name, String adjective){
        super(name);
        this.adjective = adjective;
        introduceYourself();
    }

    /** Accessor of the popularity attribute ...4 lines */
    public int getPopularity(){
        return this.popularity;
    }
    /** Accessor of the adjective attribute ...4 lines */
```

The Cowboy **is a** Character
Specialization = **Inheritance**

The Cowboy **has a** Gun
possession = **Composition**

final keyword

[final] means « Something that cannot be changed/redefined »

final Classes : Cannot be inherited, it is the last leaf of a class diagram (tree)

final methods : Cannot be overriden (completed and/or modified)

final variables : Once initialized, value cannot be modified (e.g constants)

PART 9

Elements

Collections

A collection represents a group of objects, known as its *elements*.

Some collections allow duplicate elements and others do not.

Some are ordered and others unordered

Arrays

- An array allows to manage a list of instances or a list of data of the same type

- Syntax:
  ClassName [ ] variableName;

or

  primitiveDataType [ ] variableName;

- **Size of an array is given only once** during its initialization and **can't be changed** during the execution of the program
- Syntax for initializing the size of an array:
  variableName = new ClassName[size_of_the_array];
- Syntax for accessing a specific value in an array:
  variableName[index_of_specific_value]
- The index of the first value is always equal to 0 (zero)

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Primitive Arrays

```
int[] arrayOfInt = {1, 2, 3, 4, 5};
int total = 0;

// suffix length allows to know the size of an array
for(int index=0; index<arrayOfInt.length; index++)
    total+=arrayOfInt[index];

 System.out.println("Total of all values in arrayOfInt: "+total);
```

Arrays of objects

```
SimpleDate[] arrayOfSimpleDate;

 // Initialization of the size of the array
 arrayOfSimpleDate = new SimpleDate[5];

 // Initialization of the content of the array
 for(int index=0; index<arrayOfSimpleDate.length; index++)
     arrayOfSimpleDate[index] = new SimpleDate();

 // loop to see all the values in the array
 for(int index=0; index<arrayOfSimpleDate.length; index++)
     arrayOfSimpleDate[index].writeSimpleDate();
```

Arrays of objects

- In Java, all classes inherit from the **generic class Object**
- This allows to create **“generic” arrays** which could manipulate class instances and primitive data types through the use of wrappers
- Example:

```
Object[] genericArray = new Object[3];

genericArray[0] = new SimpleDate();
genericArray[1] = new String("Hello World");
genericArray[2] = new Integer(12);

// loop to see all the values in the array
for(int index=0; index<genericArray.length; index++) {
System.out.println(genericArray[index].toString());
}
```

```
mpleDate.HelloWorld  main
- HelloWorld (run)
run:
18/9/2017
Hello World
12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Sorting an array

- Arrays of primitive types could be sorted using the following methods:
  - **void Arrays.sort(arrayToSort[])**: Sorts the elements of the array of a primitive type into ascending order using their natural ordering.
  - **void Arrays.sort(arrayToSort[], int fromIndex,int toIndex)**: Sorts the elements arrayToSort[from]...arrayToSort[to-1] of a primitive type into ascending order.

```
int[] intTable={2, 5, 1, 3, 4};
Arrays.sort(intTable);

// Will print "1 2 3 4 5 "
        for(int index=0;index<intTable.length;index++) {
            System.out.print(intTable[index]+" ");
        }
```

impleDate.HelloWorld > main >

t - HelloWorld (run)

```
run:
1 2 3 4 5 BUILD SUCCESSFUL (total time: 0 seconds)
```

Sorting an array

- Arrays of primitive types could be sorted using the following methods:
  - **void Arrays.sort(arrayToSort[])**: Sorts the elements of the array of a primitive type into ascending order using their natural ordering.
  - **void Arrays.sort(arrayToSort[], int fromIndex,int toIndex)**: Sorts the elements arrayToSort[from]...arrayToSort[to-1] of a primitive type into ascending order.

```
int[] intTable={2, 5, 1, 3, 4};
Arrays.sort(intTable,1,4);

// Will print "2 1 3 4 5 "
        for(int index=0;index<intTable.length;index++) {
            System.out.print(intTable[index]+" ");
        }
```

```
mpleDate.HelloWorld  main
- HelloWorld (run)
run:
2 1 3 5 4 BUILD SUCCESSFUL (total time: 0 seconds)
```

Sorting an array

- Arrays of objects could be sorted using the same type of methods as for the primitive types.
- But, as the computer could not know how to order two objects, you need to explain it how it should work.
- For that, you need to define a **Comparator**
- A comparator compare two objects (A and B) and returns:
  - **-1** if object A is less than object B
  - **0** if object A is equal to object B
  - **+1** if object A is greater than object B

Sorting an array

```java
public class User {
    private String firstName, lastName;
    private double money;

    public User(String firstName, String lastName, double money) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.money = money;
    }

    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public double getMoney() { return money; }
}
```

Sorting an array

```
public class UserComparator1 implements Comparator {

    public int compare(Object o1, Object o2) {
        User user1 = (User) o1;
        User user2 = (User) o2;

        if(user1.getLastName().compareTo(user2.getLastName())!=0) {
            return user1.getLastName().compareTo(user2.getLastName());
        }
        else {
            return user1.getFirstName().compareTo(user2.getFirstName());
        }
    }
}
```

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Sorting an array

```
public class UserComparator2 implements Comparator {

    public int compare(Object o1, Object o2) {
        User user1 = (User) o1;
        User user2 = (User) o2;

        if(user1.getLastName().length() > user2.getLastName().length()) {
            return -1;
        }
        if(user1.getLastName().length() < user2.getLastName().length()) {
            return 1;
        }

        if(user1.getFirstName().length() > user2.getFirstName().length()) {
            return -1;
        }
        if(user1.getFirstName().length() > user2.getFirstName().length()) {
            return 1;
        }

        return 0;
    }
}
```

Sorting an array

```
public class UserComparator3 implements Comparator {

    public int compare(Object o1, Object o2) {
        User user1 = (User) o1;
        User user2 = (User) o2;

        if(user1.getMoney() > user2.getMoney()) {
            return -1;
        }

        if(user1.getMoney() < user2.getMoney()) {
            return 1;
        }

        return 0;
    }
}
```

```java
public class SortingAnArrayOfObject {

    public static void printArray(User[] array){
         for(int index=0;index<array.length;index++)
            System.out.print(index + ": " + array[index].getLastName()+" ");
        System.out.println("");

    }

    public static void main(String[] args) {
        User user1 = new User("George","Washington",100);
        User user2 = new User("Albert","Einstein",10000);
        User user3 = new User("Amazing","RichBuddy",100000);

        User[] userList={user1,user2,user3};

        Arrays.sort(userList,new UserComparator1());  printArray(userList);
        Arrays.sort(userList,new UserComparator2());  printArray(userList);
        Arrays.sort(userList,new UserComparator3());  printArray(userList);
    }
}
```

t - HelloWorld (run)

```
run:
0: Einstein 1: RichBuddy 2: Washington
0: Washington 1: RichBuddy 2: Einstein
0: RichBuddy 1: Einstein 2: Washington
BUILD SUCCESSFUL (total time: 0 seconds)
```

Enumeration

- An enumerated type is a type for which you give all the values in a short list
- Syntax:
  enum Type_Name {FIRST_VALUE, SECOND_VALUE, … , LAST_VALUE}
- All the values are considered as constants and use the same naming convention
- Enumerated type is always placed **outside** of all methods **like named constants**
- A variable of an enumerated type could only have a **value** of the **short list** or the value **null**

Enumeration

Some functions you could use with an enumerated type:

- **boolean equal(Any_Value_Of_An_Enumerated_Type)**: Returns true if the argument is the same value as the calling instance value
- **String toString()**: Returns the calling instance value as a String
- **int ordinal()**: Returns the position of the calling instance value in the list of enumerated type values (first position equal to 0)
- **int compareTo(Any_Value_Of_An_Enumerated_Type)**: Returns a negative value if the calling instance value precedes the argument in the list of enumerated type values, returns 0 if the calling instance value equals the argument and returns a positive value if the argument precedes the calling instance value
- **EnumeratedType[] values()**: Returns an array whose elements are the values of the enumerated type in the order in which they are listed in the definition of the enumerated type
- **EnumeratedType valueOf(String name)**: Returns the enumerated type value with the specified name (must be an exact match or returns null)

Enumeration

```
public enum UsualWorkDay {
    MONDAY ("Monday, I am motivated."),
    TUESDAY ("Tuesday, usually not my best day."),
    WEDNESDAY ("Wednesday, mid-Week stress-free and relaxed."),
    THURSDAY ("Thursday, Almost done, usually a tiring day."),
    FRIDAY ("Friday, energised, can't wait to start the Week-End.");

    private String howIFeel;

    private UsualWorkDay(String feeling) {
        howIFeel = feeling;
    }
    public String toString () {
        return howIFeel;
    }

}
```

Enumeration

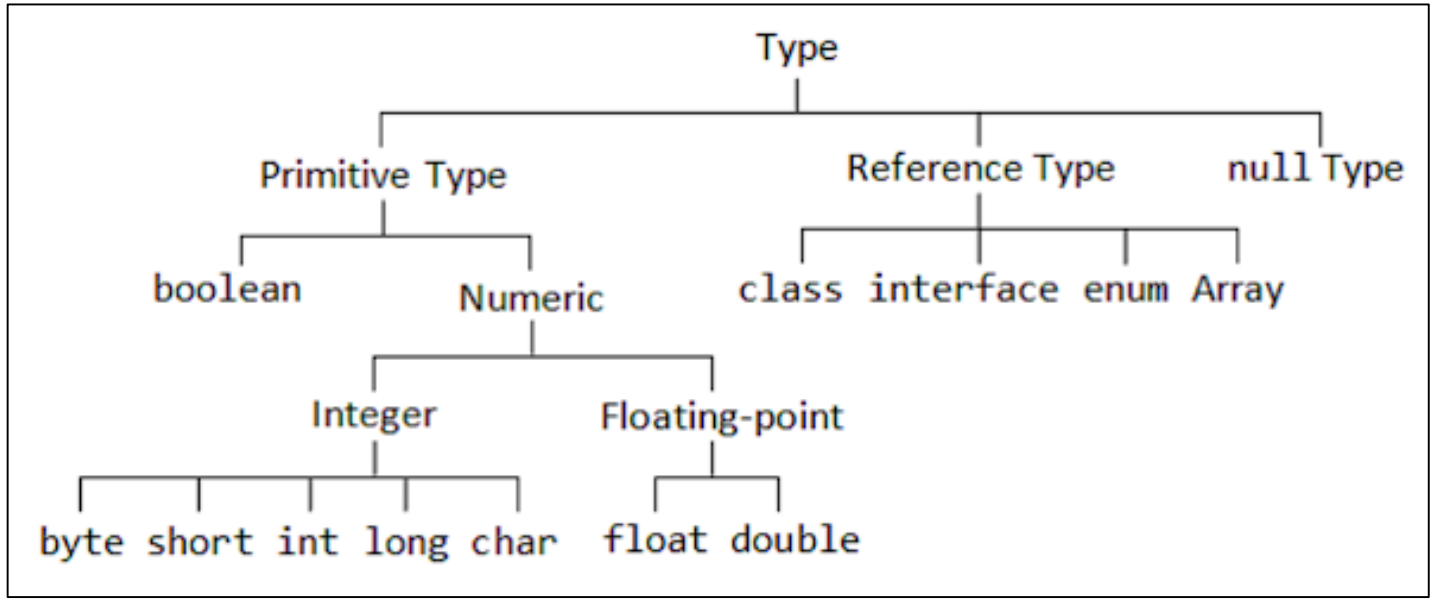
```
    public static void main(String ... args){

        UsualWorkDay today = UsualWorkDay.THURSDAY;
        System.out.println(today);
        System.out.println(today.ordinal());
```

simplePackage.SimpleDate2

ut - SimpleDate (run)

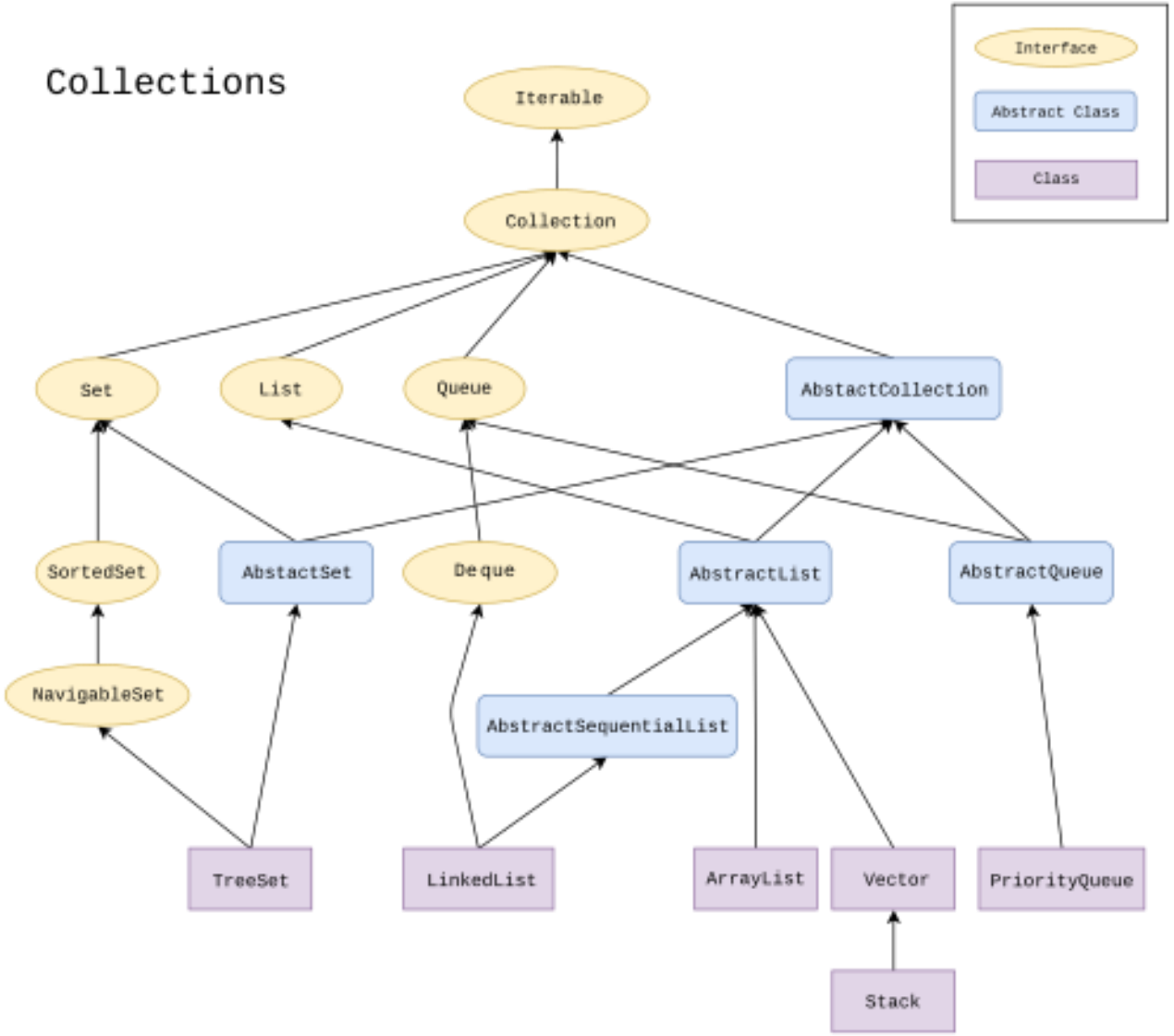
```
run:
Thursday, almost done, usually a tiring day.
3
BUILD SUCCESSFUL (total time: 0 seconds)
```

SUMMARY OF ALL DATA TYPES IN JAVA



JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Collections



Lists

A list is an ordered collection (also known as a sequence).

The user of this interface has precise control over where in the list each element is inserted

The user can:

access elements by their integer index (position in the list)
and search for elements in the list.

LinkedList and **ArrayList** are two different implementation of the List Interface

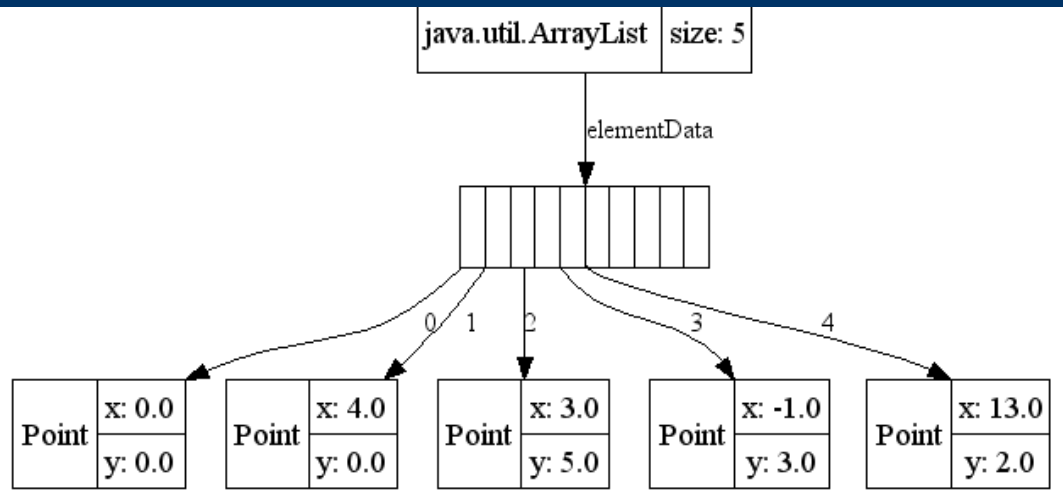
LinkedList : Double-Linked List

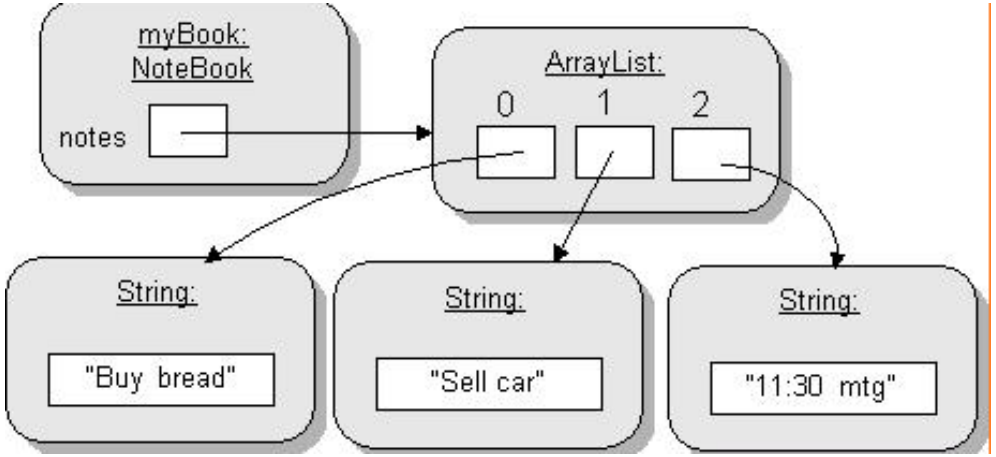
ArrayList : Dynamically resizing array

ArrayLists

Improved array (dynamic size)
References to Objects are stored continuously
Objects are stored anywhere.

You can only store Object Datatype





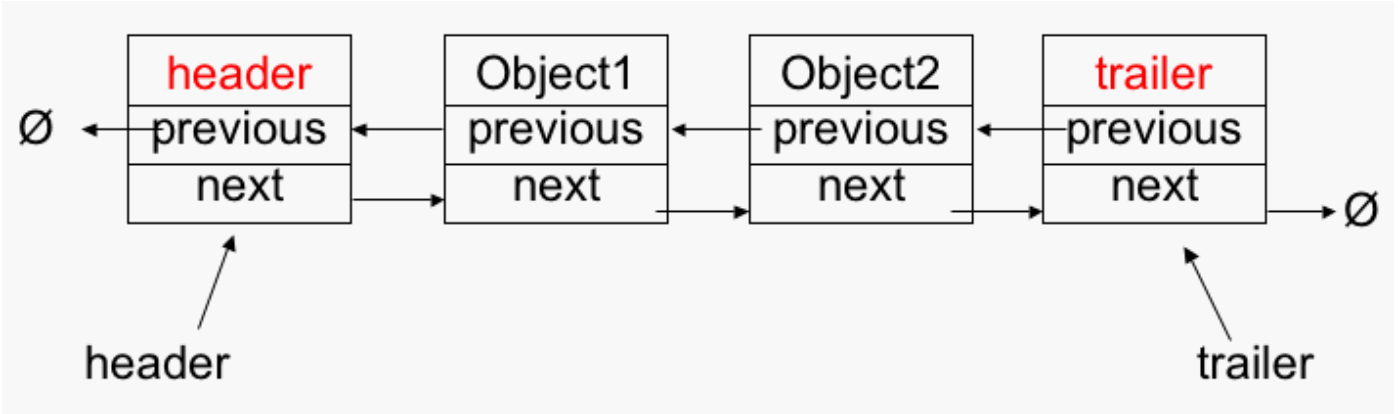
\+ Best choice when we need to access elements often (get and set methods)
\- Worst choice when you need to add/remove objects often

LinkedLists

Based on a double linked List

Objects are stored anywhere as for ArrayLists

+ Better performance for adding and removing elements (add, delete methods)
- Poorer performance for accessing and modifying elements (get, set methods)



ArrayList Class

- Syntax: **ArrayList <BaseType>** variableName **= new ArrayList <BaseType> (**size**)**
- **ArrayList** is a class in the standard Java libraries
- ArrayList allows to create objects which act like arrays that **can grow and shrink** in length while your program is running
- **BaseType** could be any Class
- ArrayList is a class of the package **java.util**

```
ArrayList <Character> myFirstList = new ArrayList <Character> ();
ArrayList <Cowboy> mySecondList = new ArrayList <Cowboy> ();
ArrayList <Integer> myThirdList = new ArrayList <Integer> ();

List <ObjectClass> myFourthList = new ArrayList <Object> ();
…
```

ArrayList Class

Methods:

- **boolean add(Instance e)**: Appends the specified instance to the end of the list
- **void add(int index, Instance e)**: Appends the specified instance at the specified position
- **void clear()**: Removes all the elements from the list
- **boolean contains(Instance e)**: Returns true if the instance e is in the list
- **base_type get(int index)**: Returns the instance at the specified position
- **int indexOf(Instance e)**: Researches the first occurrence of the instance e in the list
- **boolean isEmpty()**: Returns true if list is empty
- **int lastIndexOf(Instance e)**: Returns the index of the last occurrence of the specified instance in this list
- **base_type remove(int index)**: Removes the element at the specified position in the list
- **boolean remove(Instance e)**: Removes the first occurrence of the instance e in the list
- **base_type set(int index, Instance e)**: Replaces the element at the specified position in the list with the instance e
- **int size()**: Returns the size of the list
- **base_type[] toArray()**: Returns an array containing all of the elements in the list in the correct order
- **void trimToSize()**: Trims the capacity of the ArrayList instance to be the list's current size (allows to save space)

ArrayList Class

```
public class Western {
    public static void main(String[] args) {


        //Creation of a List able to store 10 elements
        ArrayList <Character> myFirstList = new ArrayList<Character>();

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        Robber butch = new Robber("Butch Cassidy", "Wicked");
        Lady sam = new Lady("Samantha", "Pink");

        myFirstList.add(sam);
        myFirstList.add(luke);
        myFirstList.add(butch);
```

Sorting a List

- For sorting an ArrayList, you can use two methods:
  void Collections.sort(myArrayList): sorts the ArrayList
  void Collections.sort(myArrayList, Collection.reverseOrder): sort the ArrayList in the reverse order

- All the objects inside the ArrayList should be **Comparable**.

- To shuffle the content of an ArrayList, you could use the method:
  - **void Collections.shuffle(myArrayList)**

Sorting a List

```
public abstract class Character implements Comparable{
    protected String name;

    /** Character constructor, it is not possible ...5 lines */
    public Character(String name){...3 lines }

    /** Accessor of the name attribute ...4 lines */
    public String getName(){...3 lines }

    /** method that allows the character to talk and ...5 lines */
    public void talk(String say){...3 lines }

    /** Each time a character is created, he/she has ...4 lines */
    public void introduceYourself(){...3 lines }

    public abstract String whoAreYou();

    public int compareTo(Object other) {
        Character user1 = (Character) other;
        return this.getName().compareTo(user1.getName());
    }

}
```

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

```java
public class Western {
    public static void main(String[] args) {


        //Creation of a List able to store 10 elements
        ArrayList <Character> myFirstList = new ArrayList<Character>();

        Cowboy luke = new Cowboy("Lucky Luke","Brave");
        Robber butch = new Robber("Butch Cassidy", "Wicked");
        Lady sam = new Lady("Samantha", "Pink");

        myFirstList.add(sam);
        myFirstList.add(luke);
        myFirstList.add(butch);

        System.out.println("");
        printList(myFirstList);

        Collections.sort(myFirstList);

        System.out.println("");
        printList(myFirstList);

        System.out.println("");
    }

    public static void printList(ArrayList<Character> liste){
        for (Character c : liste)
            System.out.print(c.getName() + " ... ");
    }
}
```

```
western.Western
t - Western (run)
run:

Samantha ... Lucky Luke ... Butch Cassidy ...
Butch Cassidy ... Lucky Luke ... Samantha ...
BUILD SUCCESSFUL (total time: 0 seconds)
```

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

PART 10

Abstraction