Topic: Simple Inheritance.

OOP concepts involved: Inheritance, Encapsulation, Abstraction, Polymorphism.

Programming generic concepts involved: Functions, Data Types, Variables, Access Modifiers.

➢ Theoric introduction

SIMPLE INHERITANCE: DERIVED CLASSES

Inheritance is an important tool we can use when programming in Java. When you want to create a new class and there is already a class that includes some of the code and behavior that you want, you can just derive a new class from that existing class.

Inheritance is deeply integrated into Java itself. All classes in Java, including the ones that you create, all inherit from a common class called *Object.java*. Many classes inherit directly from that class, while other inherit from those subclasses and so on, forming a hierarchy of classes.

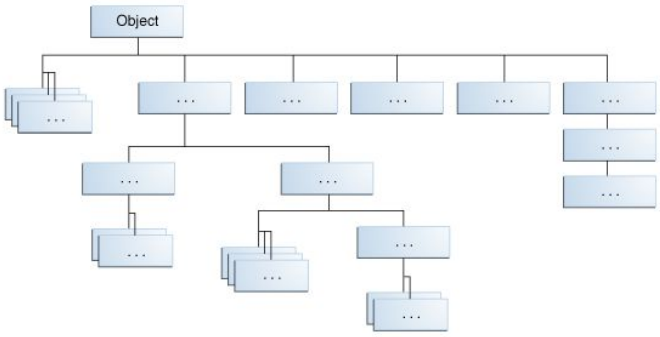


fig 1. Example of inheritance hierarchy tree

One thing to keep in mind, is that Java does not support multiple Inheritance. What this means, is that a single class cannot extend two classes.

Inheritance is done with the keyword Extends using this syntax:

```
[Modifiers] class Derived_Class extends Super_Class{}
```

A subclass will inherit all *public* and *protected* members (fields, methods and nested classes) from its superclass no matter what package the subclass is in. If the subclass is in the same package as its parent, it will also inherit all package-private members of the parent. One important observation is that Constructors are not members, so they are not inherited by subclasses, but they can still be invoked from the subclass. It's also important to note that extending an already derived class will also inherit all members from that class' superclass too.

What about private members?

When inheriting a class, we don't inherit the private members of its parent class. However, if this superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.

Another way to get access to those private members is by inheriting a nested class. Since nested classes have access to all private members of its enclosing class, by inheriting a nested class, we get indirect access to all of the private members of the superclass.

What can you do with those inherited members?

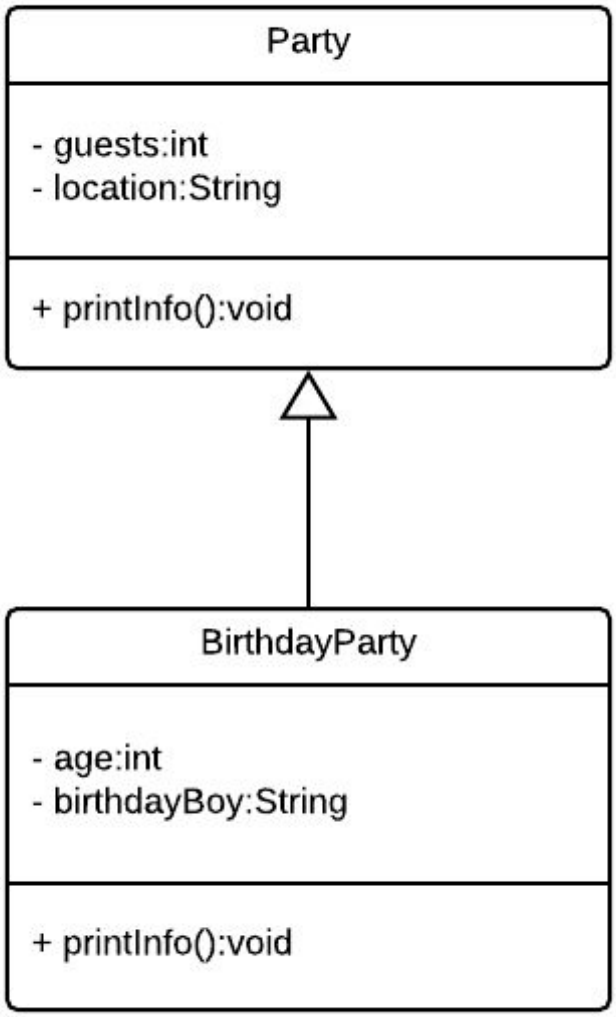
Subclasses allow us to do a variety of things to inherited members to mold them to our specific needs.

- Inherited fields can be used directly, just like any other fields
- You can declare new fields in the subclass with the same name as an already existing field in the parent class, thus *hiding* it. *(This is not recommended)*
- You can declare new fields in the subclass that are not in the superclass.
- The inherited methods can be used directly as they are.
- You can write a new instance method in the subclass that has the same signature as one in the superclass, thus *overriding* it.
- You can write a new static method in the subclass that has the same name signature as one in the superclass, thus *hiding* it.
- You can declare new methods in the subclass that are not in the superclass.
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword **super**.

➢ **Statement**

Create a PartyCreator class that allows its user to create Party invitations for different types of parties. Make use of inheritance to save time and reuse code when creating the different party types and its methods.

➢ **Class design (UML)**



➢ Program Code

Party.java

```
public class Party {
	private int guests;
	private String location;

	public Party(int guests, String location) {
		this.location = location;
		this.guests = guests;
	}

	public void printInfo() {
		System.out.println("YOU ARE INVITED TO MY PARTY");
		System.out.println("Guests:   " + this.guests);
		System.out.println("Location: " + this.location);
	}

	//Getters and Setters
	public void setGuests(int guests) {
		this.guests = guests;
	}

	public void setLocation(String location) {
		this.location = location;
	}

	public String getLocation() {
		return this.location;
	}

	public int getGuests() {
		return this.guests;
	}
}
```

BirthdayParty.java

```
public class BirthdayParty extends Party {
	private String birthdayBoy;
	private int age;

	public BirthdayParty(int guests, String location, String birthdayBoy, int age) {
		super(guests,location);
		this.birthdayBoy = birthdayBoy;
		this.age = age;
	}
```

```
	@Override
	public void printInfo() {
		System.out.println("\nWE WANT TO INVITE YOU TO " + this.birthdayBoy + "'S
BIRTHDAY PARTY");
		System.out.println("CELEBRATING " + this.age + " YEARS");
		super.printInfo();
	}

	//Getters and Setters
	public String getBBoy() {
		return this.birthdayBoy;
	}

	public int getAge() {
		return this.age;
	}

	public void setBBoy(String birthdayBoy) {
		this.birthdayBoy = birthdayBoy;
	}

	public void setAge(int age) {
		this.age = age;
	}

}
```

PartyCreator.java

```
import java.util.Scanner;

public class PartyCreator {

	public static void main(String[] args) {
		int op;
		int guests;
		String location;
		Party party;
		Scanner sc = new Scanner(System.in);

		System.out.println("What kind of party do you want?");
		System.out.println("[1] Regular Party");
		System.out.println("[2] Birthday Party");
		System.out.println("Selection: ");
		op = Integer.parseInt(sc.nextLine());
		System.out.println("How many guests do you want?: ");
		guests = Integer.parseInt(sc.nextLine());
		System.out.println("Where is this party going to be?: ");
```

```
                location = sc.nextLine();

                if (op == 1) {
                        party = new Party(guests,location);
                } else {
                        String bboy;
                        System.out.println("Who is the birthday boy?: ");
                        bboy = sc.nextLine();
                        System.out.println("How many years are you celebrating?: ");
                        party = new
BirthdayParty(guests,location,bboy,Integer.parseInt(sc.nextLine()));     //Since
BirthdayParty is extending Party, we can cast it into a Party object type
                }

                party.printInfo();
        }
}
```

➢ Program execution

During execution, we can select which kind of party we want to create, and depending on the selection, we input the data needed for that kind of party. At the end, we get an invitation created based on the information we provided.

```
What kind of party do you want?
[1] Regular Party
[2] Birthday Party
Selection:
2
How many guests do you want?:
15
Where is this party going to be?:
My house
Who is the birthday boy?:
Omar Osuna
How many years are you celebrating?:
21

WE WANT TO INVITE YOU TO Omar Osuna'S BIRTHDAY PARTY
CELEBRATING 21 YEARS
YOU ARE INVITED TO MY PARTY
Guests:   15
Location: My house
```

➢ Conclusions

Inheritance is an extremely useful tool that helps to reduce code duplication. With the code written in the parent class, we no longer have to worry about rewriting it on multiple child classes that have the same or a similar behavior. In this way, Inheritance saves us both time and resources when coding.