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.

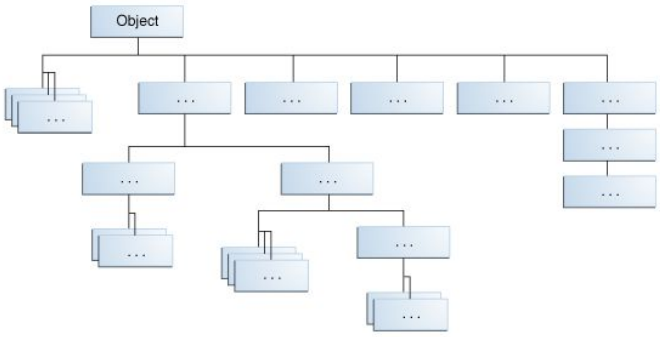


fig 1. Example of inheritance hierarchy tree

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?

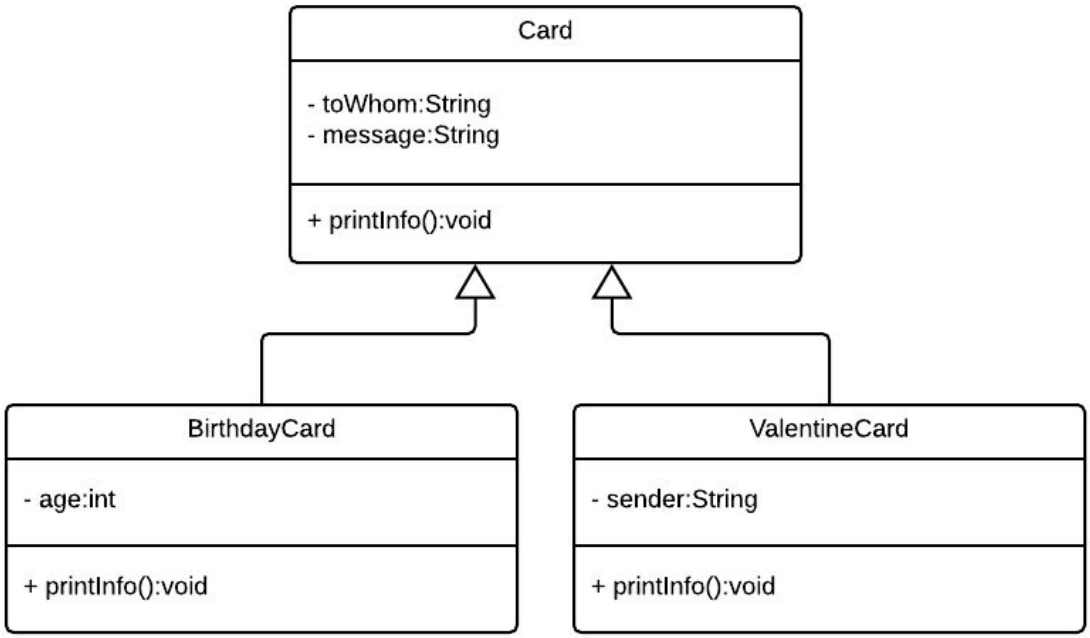
Subclasses allow us to do a variety of things to inherited members to mold them to our specific needs.

- 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 CardCreator class that allows a user to create custom messages to send on special dates. Make use of inheritance to create multiple card templates for some of the most popular holidays.

➢ **Class design (UML)**



➢ **Program Code**

Card.java

```
public class Card {
	private String message;
	private String toWhom;

	public Card(String toWhom, String message) {
		this.message = message;
		this.toWhom = toWhom;
	}
```

```
	public Card(Card card) {
		this.message = card.message;
		this.toWhom = card.toWhom;
	}

	public void printInfo() {
		System.out.println("\n\nTO: " + this.toWhom);
		System.out.println(this.message);
	}

	//Getters and Setters
	public String getMessage() {
		return this.message;
	}

	public String getDestinatary() {
		return this.toWhom;
	}

	public void setMessage(String message) {
		this.message = message;
	}

	public void setDestinatary(String toWhom) {
		this.toWhom = toWhom;
	}
}
```

BirthDayCard.java

```
public class BirthDayCard extends Card {
	private int age;

	public BirthDayCard(String toWhom, String message, int age) {
		super(toWhom,message);
		this.age = age;
	}

	@Override
	public void printInfo() {
		System.out.println("\n\nBIRTHDAY WISHES " + this.getDestinatary() + " ON YOUR
" + this.getAge() + "TH BIRTHDAY!");
		System.out.println(this.getMessage());
	}

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

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

}
```

ValentineCard.java

```
public class ValentineCard extends Card {
	private String sender;

	public ValentineCard(String toWhom, String message, String sender) {
		super(toWhom,message);
		this.sender = sender;
	}

	@Override
	public void printInfo() {
		System.out.println("\n\nFor a dear friend on this valentine day!");
		System.out.println("To: " + this.getDestinatary());
		System.out.println("From: " + this.getSender());
		System.out.println(this.getMessage());
	}

	public void setSender( String sender ) {
		this.sender = sender;
	}

	public String getSender() {
		return this.sender;
	}

}
```

CardCreator.java

```
import java.util.Scanner;

public class CardCreator {

	public static void main(String[] args) {
		int op;
		String message;
		String toWhom;
		Card card;
		Scanner sc = new Scanner(System.in);
```

```
                System.out.println("What kind of party do you want?");
                System.out.println("[1] Regular Card");
                System.out.println("[2] Birthday Card");
                System.out.println("[3] Valentine Card");
                System.out.println("Selection: ");
                op = Integer.parseInt(sc.nextLine());
                System.out.println("Who is the recipient?: ");
                toWhom = sc.nextLine();
                System.out.println("What's the message you want to send?: ");
                message = sc.nextLine();

                if (op == 1) {
                        card = new Card(toWhom,message);
                } else if (op == 2) {
                        System.out.println("How many years are we celebrating?: ");
                        card = new BirthDayCard(toWhom,message,Integer.parseInt(sc.ne
xtLine()));
                } else {
                        System.out.println("Who is sending this card?: ");
                        card = new ValentineCard(toWhom,message,sc.nextLine());
                }

                card.printInfo();
        }
}
```

➢ Program execution

During program execution, we can create different types of card and input the info needed for the type wanted. This is a simplified example, but it can be expanded to include more types of cards or even more detailed card information.

```
What kind of party do you want?
[1] Regular Card
[2] Birthday Card
[3] Valentine Card
Selection:
3
Who is the reciever?:
Andre
Whats the message you want to send?:
Thank you for being a great friend all this time, you are the best!
Who is sending this card?:
Omar


For a dear friend on this valentine day!
To: Andre
From: Omar
Thank you for being a great friend all this time, you are the best!
```

➢ 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.