Java Programming I

Session 6

EXTENDING CLASSES AND INHERITANCE ACCESS QUALIFIERS

Juan Carlos Moreno - UCLA Ex

Agenda

- **Basic OOP**
- **Super**
- **Instance of**
- **Abstract**
- **Interfaces**
- **Final**
- **Coding exercises**

Basic OOP Theory

Object oriented Programming

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

restrict/hide the implementation details from users

```
public class Encapsulation{
    private String email;

    public String getEmail(){
        return email;
    }

    public void setEmail(String new_email){
        if (!new_email.contains("."))
            return;

        if (!new_email.contains("@"))
            return;

        email = new_email;
    }
}
```

{ Restricts read/write access

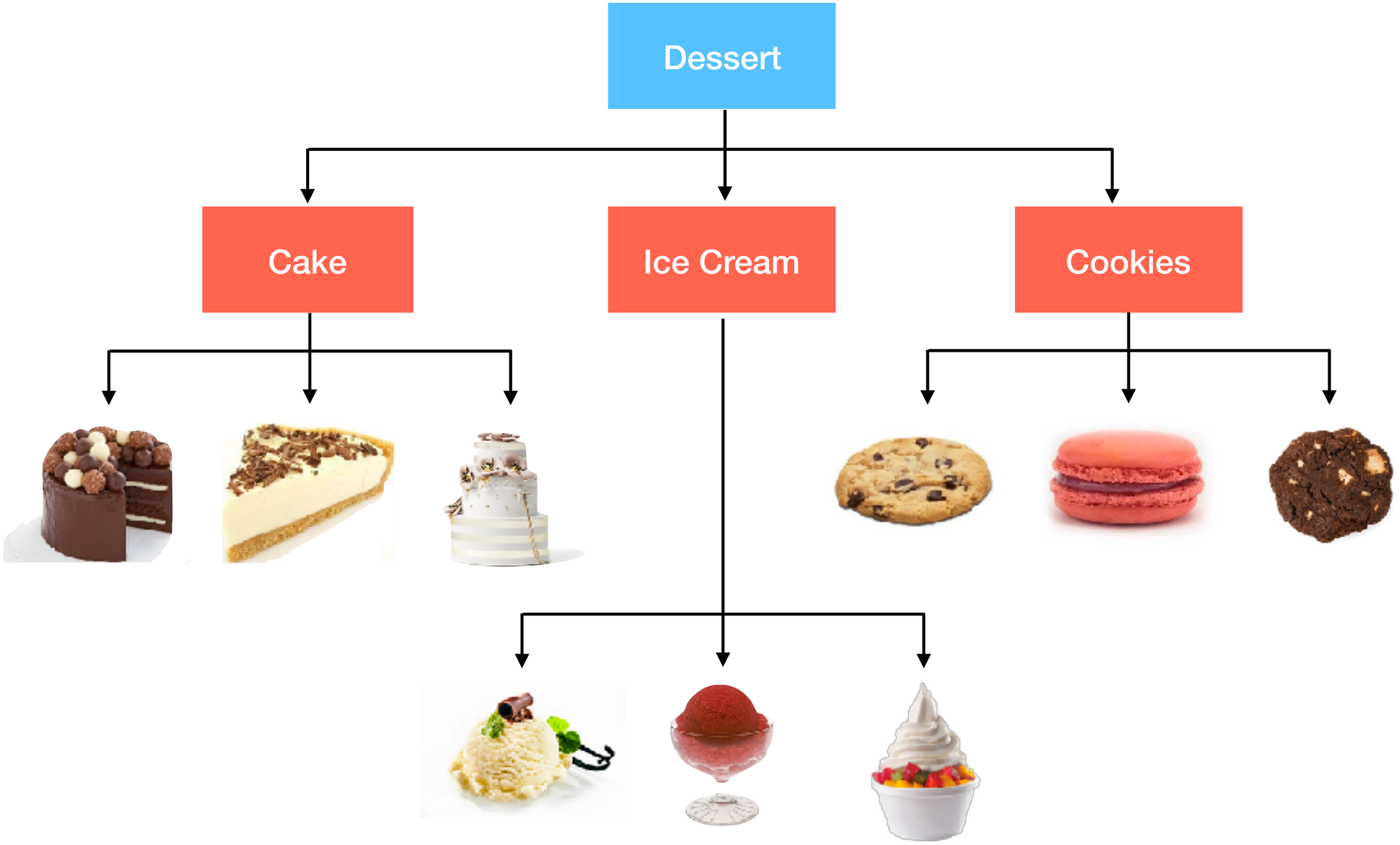
Polymorphism

capability of a method to do different things based on the object that it is acting upon

```
class Polymorphic{
    void demo (int a){
        System.out.println ("a: " + a);
    }

    void demo (int a, int b){
        System.out.println ("a and b: " + a + "," + b);
    }

    double demo(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
```

Inheritance

calling the parent class



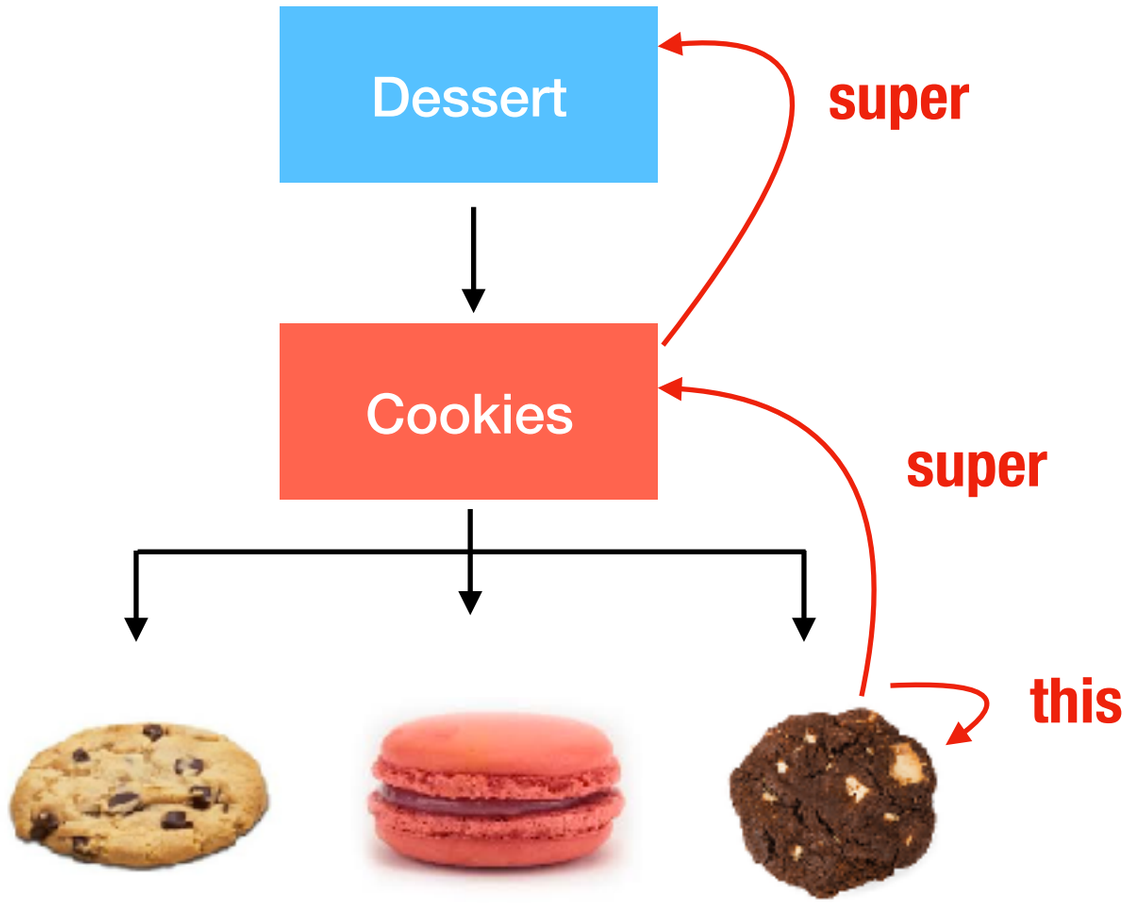
Inheritance

Allowing a class to inherit properties and methods from other classes

```
class Vehicle {
    String color;
    int speed;
    int size;

    void attributes() {
        System.out.println("Color : " + color);
        System.out.println("Speed : " + speed);
        System.out.println("Size : " + size);
    }
}

class Car extends Vehicle {
    int CC;
    int gears;

    void attributescar() {
        // The subclass refers to the members of the superclass
        this.attributes();
        System.out.println("CC of Car : " + CC);
        System.out.println("No of gears of Car : " + gears);
    }
}
```

Super

calling the parent class



Super

calling the parent class

```
package edu.ucla.ex.java.summer;

public class Cookie {
    private Shape shape;
    private Ingredient ingredients[];
    public String flavor_name;

    public Cookie(Ingredient new_ingredients, CookieCutter cutter){
        this.ingredients = new_ingredients;
        this.mix();
        this.shape = cutter.cut();
    }

    public void mix() {
        int times = 5;
        for (int fold = 0; fold < times; fold++) {
            Random rand = new Random();
            for (int i = 0; i < this.ingredients.length; i++) {
                int random_element = rand.nextInt(this.ingredients.length);
                Ingredient to_swap = this.ingredients[i];
                this.ingredients[i] = this.ingredients[random_element];
                this.ingredients[random_element] = to_swap;
            }
        }
    }
}
```

Super

calling the parent class

```
public class ChocolateChipCookie extends Cookie{

    private Ingredient choc_chips[];

    public void mix() {
        super.mix();
        int len = choc_chips.length + this.ingredients;
        Ingredient[] new_ingredients = new Ingredient[len];

        // Add cookies to the end
        for (int i=0; i<len; i++) {
            if (i < this.ingredients.length){
                new_ingredients[i] = this.ingredients[i];
            } else {
                int choc_index = i - this.ingredients.length;
                new_ingredients[i] = this.choc_chips[choc_index];
            }
        }
    }
}
```

instanceof

Are all of these cookies?

```
if (my_oreo instanceof Cookie){
    CookieMonster.eat(my_oreo);
}
```

Abstract class

Not quite there yet

Abstract class

calling the parent class

```
public abstract class Cookie {
    private Shape shape;
    private Ingredient ingredients[];
    public String flavor_name;
    public Oven oven;

    public Cookie(Ingredient new_ingredients, CookieCutter cutter){
        oven = Oven.singletonInstance();
        this.ingredients = new_ingredients;
        this.mix();
        this.shape = cutter.cut();
        this.bake();
    }

    public void mix() {
        ....
        ....
    }

    abstract void decorate();
}
```

Interface

Just the rules

```
public interface Chocolate

public class ChocolateCookie extends Cookie implements Chocolate
```

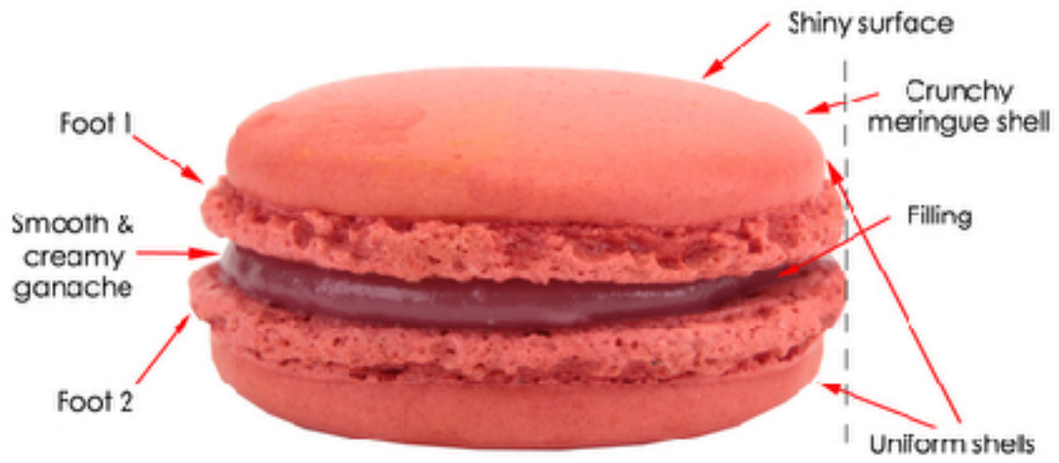
class

\+

Interface

Final

Don’t mess with perfection



Final prevents the class from being subclassed or overridden

```
public final class Macaron extends Cookie {
   // Can’t be inherited
}

public class MySecretCookie extends Cookie {
   // Can be inherited
   private final  Int folds = 10;  // Can’t change this value

   public final void mix() {
        // My secret method can’t be changed
   }
}
```

Shapes

Simple inheritance

Using the following base as a root parent object using the OOP principles, write classes that represent a Triangle, Rectangle and Circle all using the shape interface. Both, triangle and rect should use both Polygon class and shape interface
Google Area and Perimeter formulas for such shapes and include them as part of your implementation.

Submit the completed Triangle.java, Rectangle.java and Circle.java.

```
public abstract class Polygon{
    public long sides[];

    public Polygon(long ...side_lengths) throws
InstantiationException {
        if (side_lengths.length>this.sideCount())
        {
            throw new InstantiationException();
        }

        this.sides = new long[this.sideCount()];
        for(int i=0; i<side_lengths.length; i++){
            this.sides[i] = side_lengths[i];
        }
    }

    public abstract int sideCount(); // Returns 3 for
triangle, 4 for rectangle

}
```

```
public interface Shape {

    public double getPerimeter();

    public double getArea();
}
```