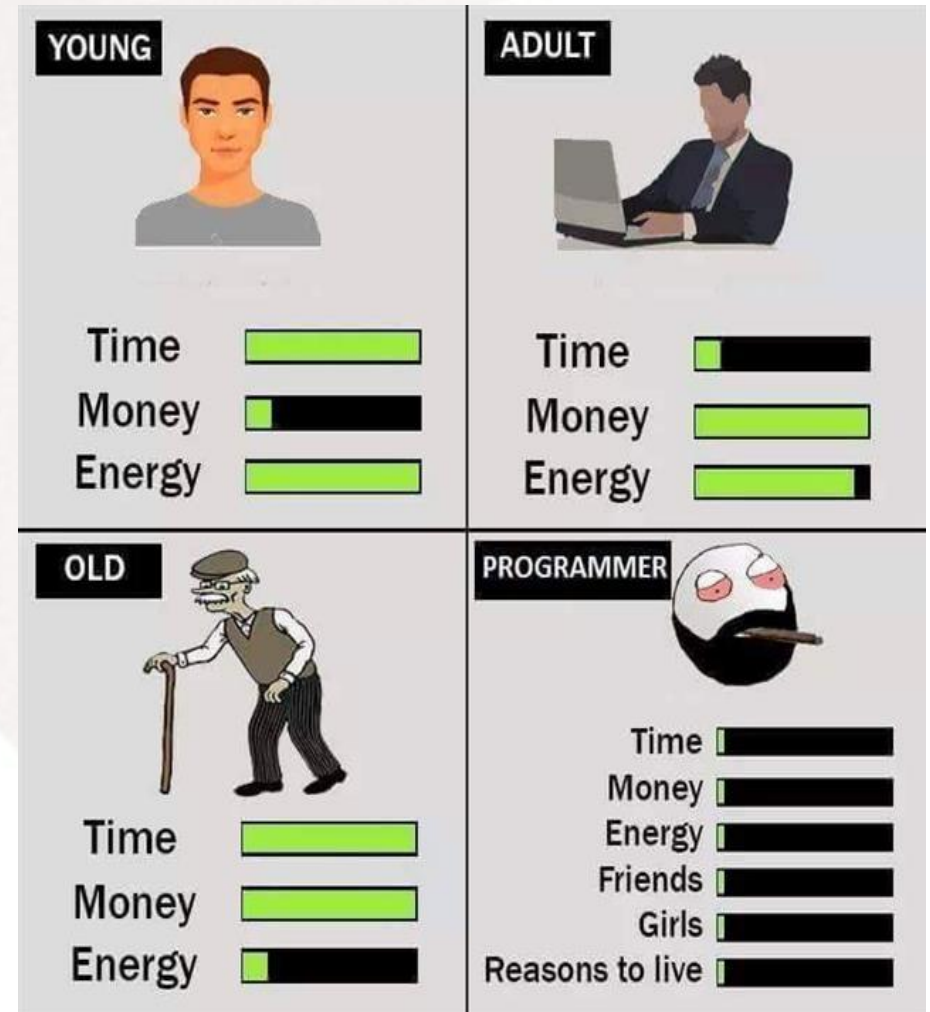
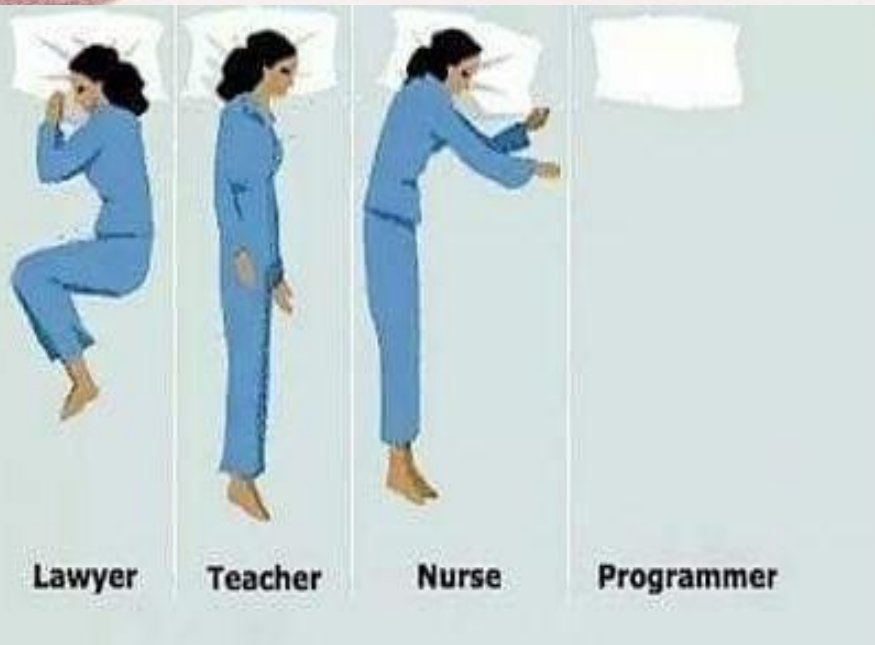


Chapter 4 - Class in java



What's a class?

A class is like a function that allow you to do a specific action. We have already seen 2 classes in the previous slides(Scanner and String)

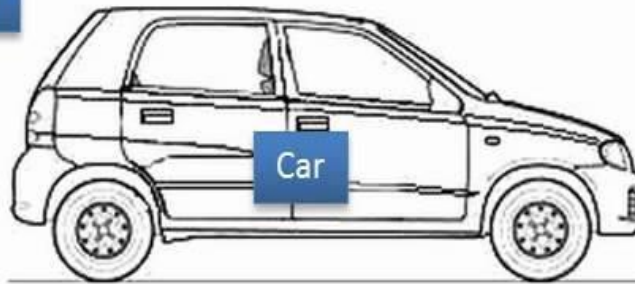
We can create our own classes to do something we want

What's a class - example

What is a Class?

A class is the blueprint from which individual objects are created.

Class



```
1 public class Car
2 {
3     private String brand = null;
4     private String model = null;
5     private String color = null;
6
7     public String getBrand()
8     {
9         return brand;
10    }
11
12    public void setBrand(String brand)
13    {
14        this.brand = brand;
15    }
16
17    public String getModel()
18    {
19        return model;
20    }
21
22    public void setModel(String model)
23    {
24        this.model = model;
25    }
26
27    public String getColor()
28    {
29        return color;
30    }
31
32    public void setColor(String color)
33    {
34        this.color = color;
35    }
36
37 }
```

Objects

```
Car maruthiAltoK10 = new Car();
maruthiAltoK10.setBrand("Maruthi Alto");
maruthiAltoK10.setModel("K10");
maruthiAltoK10.setColor("Orange");
```

brand = Maruthi Alto
model = K10
color = Orange



```
Car swift = new Car();
swift.setBrand("Swift");
swift.setModel("ZDI");
swift.setColor("Red");
```

brand = Swift
model = ZDI
color = Red



```
Car maruthiAlto800 = new Car();
maruthiAlto800.setBrand("Maruthi Alto");
maruthiAlto800.setModel("800");
maruthiAlto800.setColor("Blue");
```

brand = Maruthi Alto
model = 800
color = Blue



We can declare a class just like this,
Note: we only declare it. We haven't
created an object yet


new Operator




ClassName ObjectName;

ObjectName = new ClassName();

The “=” will make the
object associated
with the variable
name(ObjectName)



The new operator will create
the object of type
ClassName



ClassName ObjectName = new ClassName();

Example of class

```
Book b1 = new Book();
```

//Book is class name

//b1 is the object name

//b1 is also a pointer/reference to the
//object

Instance variables

Instance variable AKA (attributes)

*note: is like a global commun variables that can be used inside the class only

Example:

```
public int door;
```

```
private double book;
```

Note: instance variables are often private to prevent privacy leak

Instance variable

If we want to refer to a specific instance variable, We connect it with the object then use the “.” just like System.out.print.

Example:

c1.door;

b1.book

Object
Name



Instance variable
name

Now we have access to the object instance
variable

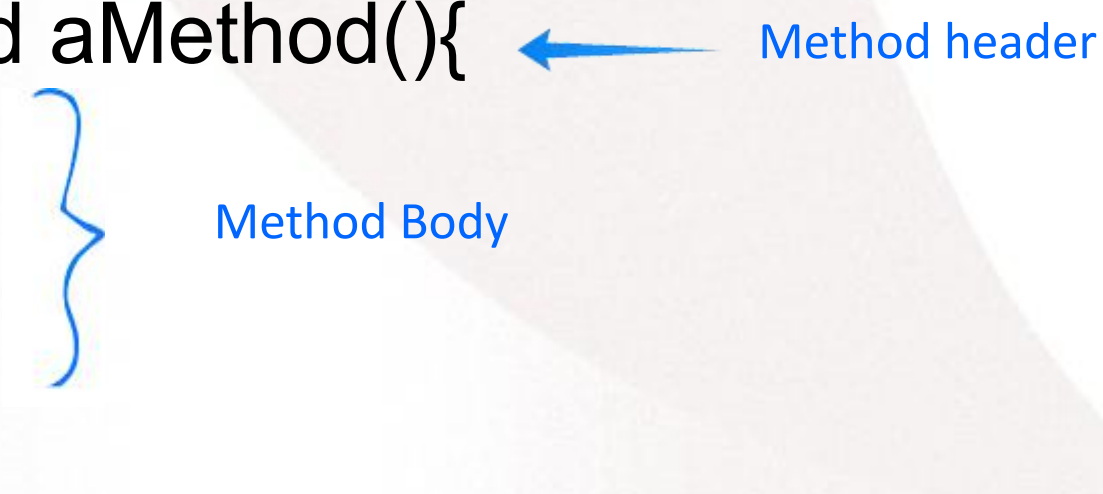
Methods

Methods: It is like a subclass of Class which we can perform specific action

Note: a method should be use to only do one job and not all

Example:

```
Public void aMethod(){  
    //code  
}
```



Method header

Method Body

Methods

We can call a method by using the following Syntax:

`objectName.MethodName;`

Example:

`c1.getPriceOfBook;`

More about methods

There are two type of Methods:

- A method that computer/perform action and **will return a value** (meaning we have to have a return)
- A method that computer/perform action and **will not return a value** (AKA: void)

The method that return nothing is called void

More about methods

For a method that does return something,

We need to specify which primitive type it return

Syntax: `public returnType methodName(paramList);`

Example:

```
public int getNumOfPeople(int people){  
    //random calculation...  
    return int;  
}
```

Note: paramList is optional and is used to pass variable for your method.

Example of a program

```
4  
5  
6 package org.businessapptester.monitoring.tcpserver.protocol;  
7
```

```
8 /**
```

```
9  * My documentation.
```

```
10 */
```

```
11 @SuppressWarnings("unused")
```

```
12 public class MyClass {
```

```
13  
14     private int intValue;
```

```
15     private String stringValue;
```

```
16  
17     // method declaration
```

```
18 public void doSomething(int intValue, String stringValue){
```

```
19     this.intValue = intValue;
```

```
20     this.stringValue = stringValue;
```

```
21     // do something with the values
```

```
22 }
```

```
23 }
```

```
24 |
```

class definition

instance variables

instance method

Constructors

A constructor or Ctor is a special method that is used to initialize all instance variable for an object, it's like a blueprint for a house

Syntax:

```
Public className(paramList){  
//code...  
}
```

Note: Ctor does not have a return type not even **void**! Also, the name must be the same as className

Example of Constructor

```
public class cars {  
  
    public cars() {  
        //random code  
    }  
  
    public cars(int a) {  
        //more randcom code  
    }  
  
}
```

Constructor with
no parameter

Constructor with
a parameter

More about Constructor

- If you do not include a constructor, java will automatically create a “invisible” default constructor so we can create objects
- If you create a constructor, then the default constructor will not be automatically create

Example of default Ctor

```
public class vehicle {  
  
    public vehicle(String name) {  
        System.out.println("==Creating vehicle with parameter==");  
        System.out.println("The name of the vehicle is: " + name + "\n");  
    }  
  
    public static void main(String [] args) {  
  
        vehicle v1 = new vehicle();  
        vehicle v2 = new vehicle("Mr.Bean");  
    }  
}
```

What's the output?

Example of default Ctor

```
public class vehicle {  
  
    public vehicle() {  
        System.out.println("==Creating vehicle without parameters==");  
        System.out.println("The vehicle doesn't have a name\n");  
    }  
  
    public vehicle(String name) {  
        System.out.println("==Creating vehicle with parameter==");  
        System.out.println("The name of the vehicle is: " + name + "\n");  
    }  
  
    public static void main(String [] args) {  
  
        vehicle v1 = new vehicle();  
        vehicle v2 = new vehicle("Mr.Bean");  
    }  
}
```

What's the output?

public vs private

public: There is no restriction of where a method/instance variable can be accessed or used

private: we cannot access a method/instance variable outside of it's class

***note:** used to prevent privacy leak

public vs private example

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier*	✓	✓	✗	✗
private	✓	✗	✗	✗

When to use public vs private?

Public:

- most method should be public
- constant can be public but highly not recommended(even though can't change value)

Private:

- instance variable should be private
- some methods, if a method is only to assist/help in program then it should be private

Local and global variables

There is no global variables in java, but there is local variable

- a variable declared inside a method is local

- if two methods have the same variable name, they are still different variable because they are local


Local variable in for loop

A variable declared in a for loop is only local to the for loop and cannot be accessed outside unless we declared the variable outside.

Examples:

```
int price = 10;
for(int i = 0; i<10; i++)
{
    price = 10;
}
System.out.println(price);
```

Cannot access variable price because it does not exist outside loop



```
for(int i = 0; i<10; i++)
{
    int price = 10;

}
System.out.println(price);
```

Parameter

A method/Constructor can have no parameter, one parameter or more than one

(Formal parameter)

Example:

```
Public void setBookInfo(double p, int y, String n){  
//code  
}
```


Parameter

When we invoke a parameter, we then need to pass the value as arguments. Order matters!

Example:

```
B1.setBookInfo(10.99, 1998, Steven_Seagal);
```

```
//this would cause a compile error based from  
//previous method
```

```
B2.setBookInfo(Jacky_Chan, 24.99, 1992);
```

Acessor and mutator

Acessor(getters): We can access the object instance variable

- We cannot change anything.
- Usually starts with a getInstanceVars

Mutators(setters): We can modify the object instance variable

- we can change the instance variable
- start with setInstanceVars

Example of acesor and mutator

```
public class question3 {  
    private int page;  
    private double price;  
  
    public void setPage(int p) {  
        page = p;  
    }  
  
    public void setPrice(int price) {  
        this.price = price;  
    }  
  
    public int getPage() {  
        return page;  
    }  
  
    public double getPrice() {  
        return price;  
    }  
}
```

Why do we
need "this" ??

toString

-toString main function is to return a string value of an object

Syntax:

```
public String toString(){  
    //code  
}
```

ToString Example

```
public class vehicle {  
  
    private int numDoors;  
    private double price;  
    private String name;  
  
    public vehicle() {  
        numDoors = 4;  
        price = 1000;  
        name = "no name";  
    }  
  
    public vehicle(int nd,  
        double p, String n) {  
        numDoors = nd;  
        price = p;  
        name = n;  
    }  
}
```

```
public String toString() {  
    return "The following are the details  
    of the vehicle: \nThe price is : "  
    + price + " there are " + numDoors +  
    " of doors and the brand is " + name  
    + "\n";  
}  
  
public static void main(String[] args) {  
  
    vehicle v1 = new vehicle();  
    vehicle v2 = new vehicle(4, 60000,  
        "Honda");  
  
    System.out.println(v1);  
    System.out.println(v2);  
    }  
}
```

What's the output of the following code?

What if we delete the toString method?

Overloading

Overloading is when two or more methods have the same name within the same class but have different signature(parameters)

Example:

```
class Dog{  
    public void bark(){  
        System.out.println("woof ");  
    }  
  
    //overloading method  
    public void bark(int num){  
        for(int i=0; i<num; i++)  
            System.out.println("woof ");  
    }  
}
```


overriding

Overriding is when two methods have the same name and parameters, but the child is overriding the parent method

Example:

Don't worry about "extend" keyword, it's inheritance which will be covered in the later chapter

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}

public class OverridingTest{
    public static void main(String [] args){
        Dog dog = new Hound();
        dog.bark();
    }
}
```

Overloading vs overriding

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

What is the meaning of “this”?

The keyword “this” has many different usage

- “this” can be used to avoid confusion between instance variable
- We can also invoke other constructor in the current class
- It can also refer to the whole object

Example of “this”

Try guessing the output

```
1 public class helloWorld {
2     private int a;
3     private int b;
4
5     public helloWorld(){
6         this(50);
7     }
8
9     public helloWorld(int a){
10        this.a = a;
11        this.b = a;
12    }
13
14    public String toString(){
15        return "A is: " + a + " B is : " + b;
16    }
17
18    public void runCode(){
19        int a = 1;
20        int b = 5;
21        System.out.println("a is: " + a + " b is: " + b);
22        System.out.println("Printing the instance variable of a and b");
23        System.out.println("a: " + this.a + " b: " + this.b);
24        System.out.println("Printing the entire object of helloWorld");
25        System.out.println(this);
26    }
27
28    public static void main(String[] args){
29        System.out.println("=====Creating h2 object=====");
30        helloWorld h1 = new helloWorld();
31        h1.runCode();
32
33        System.out.println("=====Creating h2 object=====");
34        helloWorld h2 = new helloWorld(125);
35        h2.runCode();
36    }
37 }
```

Example of “this”

```
public helloWorld(){  
    this(50);  
}
```

Will set the object instance variable to 50(basically call 1 param ctor)

```
public helloWorld(int a){  
    this.a = a;  
    this.b = a;  
}
```

Avoid confusion of instance variable

This.a will refer to the object instance variable

```
public void runCode(){  
    int a = 1;  
    int b = 5;  
    System.out.println("a is: " + a + " b is: " + b);  
    System.out.println("Printing the instance variable of a and b");  
    System.out.println("a: " + this.a + " b: " + this.b);  
    System.out.println("Printing the entire object of helloWorld");  
    System.out.println(this);  
}
```

Refer to the calling object

Example of “this” - answers

```
=====Creating h1 object=====
a is: 1 b is: 5
Printing the instance variable of a and b
a: 50 b: 50
Printing the entire object of helloWorld
A is: 50 B is :50
=====Creating h2 object=====
a is: 1 b is: 5
Printing the instance variable of a and b
a: 125 b: 125
Printing the entire object of helloWorld
A is: 125 B is :125
```


Static method

Use when you want to create a function without creating an object

Cannot use “this” since it does not have any object or instance variables

But we can invoke another static method

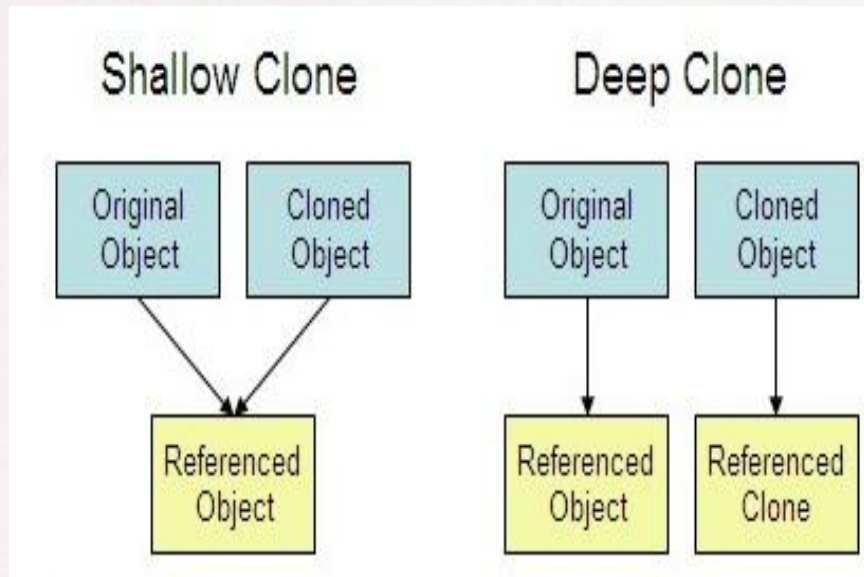
Static variables

- Static variables belong to the whole class(similar to global variables) and it does not belong to only one object
- only 1 copy of static variable per class, where as instance variable have their own copy for each objects
- all object can read/change static variable
- static method can access static variable but not instance variable
- same declaration as instance variable just add static before “data type”

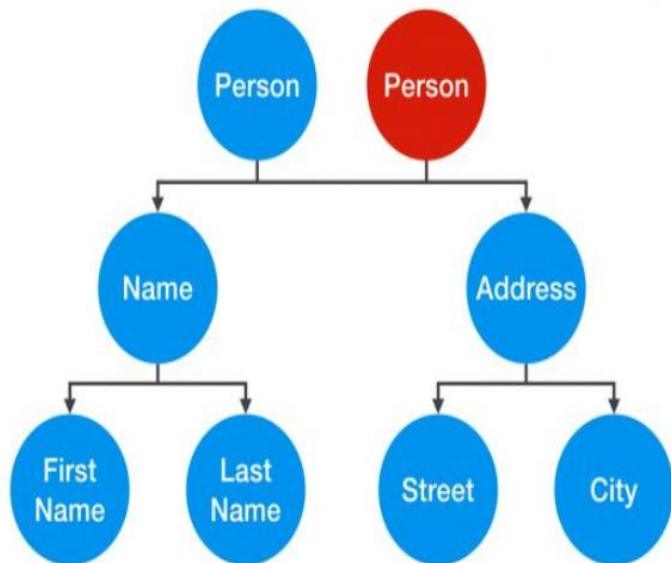
Shallow vs deep copy

- Shallow: will create an instance of object that share the same pointer/reference. Any change made will change the whole object.
- Deep: a deep copy will create a independent copy that is unrelated to the original. If any change are made, it will not affect the original copy
- Note: we can have a mixture of both

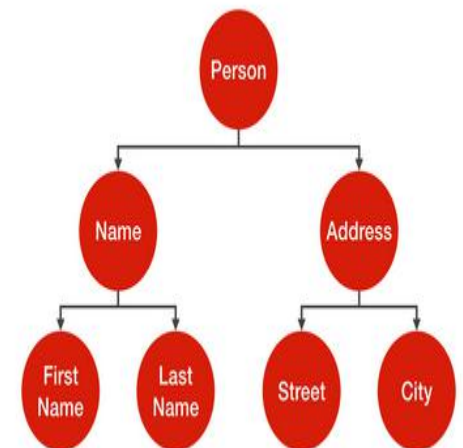
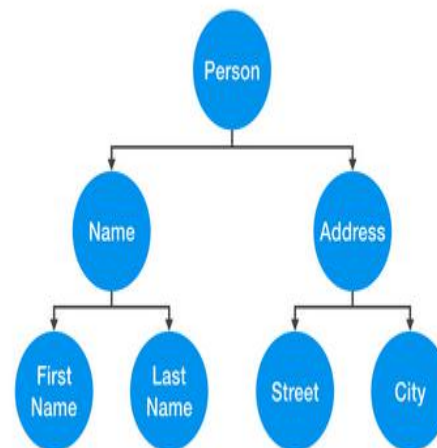
Example of shallow vs deep copy



Shallow copy



Deep Copy



Copy constructor

Sometime we want to create a duplicate object, copy ctor allows us to make a clone object that is seperate and independent

Syntax:

```
public className(className objCopy){  
    className copyObj = new className();  
    //copy instance values  
    Return copyObj;  
}
```

Copy constructor example

```
1 package born;
2
3 public class birth {
4     private String month;
5     private int year;
6
7     public birth(){
8         month = null;
9         year = 0;
10    }
11
12    public birth(String m, int y){
13        month = m;
14        year = y;
15    }
16
17    public birth(birth copy){
18        month = copy.month;
19        year = copy.year;
20    }
21
22    public void setMonth(String m){
23        month = m;
24    }
25
26    public void setYear(int y){
27        year = y;
28    }
29
30    public String getMonth(){
31        return month;
32    }
33
34    public int getYear(){
35        return year;
36    }
37
38    public String toString(){
39        return "born in : " + month + ", " + year;
40    }
41 }
```


Copy constructor example

What's the output?

```
1 package born;
2
3 public class person {
4     private int age;
5     private String name;
6     private birth born;
7
8     public person(){
9         age = 0;
10        name = null;
11        born = null;
12    }
13
14    public person(int a, String n, birth b){
15        age = a;
16        name = n;
17        born = b;
18    }
19
20    public person(person copy){
21        this.age = copy.age;
22        this.name = copy.name;
23        this.born = copy.born;
24    }
25
26    public void setAge(int a){
27        age = a;
28    }
29
30    public void setName(String n){
31        name = n;
32    }
33
34    public int getAge(){
35        return age;
36    }
```

```
37
38    public String getName(){
39        return name;
40    }
41
42    public String toString(){
43        //born will call the toString of date
44        return name + " is " + age + " year old and is born in " + born;
45    }
46
47    public void setBirthYear(int year){
48        if(born == null)
49            System.exit(0);
50
51        born.setYear(year);
52    }
53
54    public static void main(String[] args){
55        person p1 = new person(20, "john", new birth("april", 2010));
56        System.out.println("====original p1====\n" + p1);
57        person copy = new person(p1);
58
59        System.out.println("===Copy of p1===\n" + copy + "\n");
60        System.out.println("modify year to 1950");
61        copy.setBirthYear(1950);
62
63        System.out.println("===copy of p1===\n" + copy);
64        System.out.println("===original p1===\n" + p1);
65    }
66 }
```

equals methods

- Equals method are useful if we wish to compare objects. Java already have an equals method implemented, if we do not override the .equals(). Then it will only compare by reference(if object are same location memory aka (==))
- we can define our own equals method by overriding which will compare **contents** rather than location memory

Syntax:

```
public boolean equals(ClassName ObjName  
    ){ ... }
```


equals method example

1) non overridden equals()

```
1  public class equalTest {
2      private int numPage;
3      private double price;
4
5      equalTest(){
6          numPage = 0;
7          price = 0;
8      }
9
10     equalTest(int np, double p){
11         numPage = np;
12         price = p;
13     }
14
15     public String toString(){
16         return "the object has " + numPage + " of page and it cost " + price;
17     }
18
19     public static void main(String[] args){
20
21         System.out.println("=====Creating e1 object=====");
22         equalTest e1 = new equalTest();
23         System.out.println("e1 has the following properties: " + e1 + "\n");
24
25         System.out.println("=====Creating e2 object=====");
26         equalTest e2 = new equalTest();
27         System.out.println("e2 has the following properties: " + e2 + "\n");
28
29         System.out.println("testing for equal of e1 and e1");
30         System.out.println(e1.equals(e1));
31
32         System.out.println("Testing for equal of e1 and e2");
33         System.out.println(e1.equals(e2));
34     }
35 }
```

What's the output of following code without overriding equal method?

2) overridden equals

equals method example

What's the following output with equals()?

```
1 public class equalTest {
2     private int numPage;
3     private double price;
4
5     equalTest(){
6         numPage = 0;
7         price = 0;
8     }
9
10    equalTest(int np, double p){
11        numPage = np;
12        price = p;
13    }
14
15    public String toString(){
16        return "the object has " + numPage + " of page and it cost " + price;
17    }
18
19    public boolean equals(equalTest obj){
20        return ( (this.numPage == obj.numPage) && (this.price == obj.numPage) );
21    }
22
23    public static void main(String[] args){
24
25        System.out.println("=====Creating e1 object=====");
26        equalTest e1 = new equalTest();
27        System.out.println("e1 has the following properties: " + e1 + "\n");
28
29        System.out.println("=====Creating e2 object=====");
30        equalTest e2 = new equalTest();
31        System.out.println("e2 has the following properties: " + e2 + "\n");
32
33        System.out.println("testing for equal of e1 and e1");
34        System.out.println(e1.equals(e1));
35
36        System.out.println("Testing for equal of e1 and e2");
37        System.out.println(e1.equals(e2));
38    }
39 }
```

equal method example - answers

1) normal equals()

```
=====Creating e1 object=====
e1 has the following properties: the object has 0 of page and it cost 0.0

=====Creating e2 object=====
e2 has the following properties: the object has 0 of page and it cost 0.0

testing for equal of e1 and e1
true
Testing for equal of e1 and e2
false
```

2) overridden equals()

```
=====Creating e1 object=====
e1 has the following properties: the object has 0 of page and it cost 0.0

=====Creating e2 object=====
e2 has the following properties: the object has 0 of page and it cost 0.0

testing for equal of e1 and e1
true
Testing for equal of e1 and e2
true
```

null

- null means that it has “no real value”
- It's a temporary placeholder and not an object
- We cannot invoke a method that has a variable that has been initialized as null(will give NullPointerException)