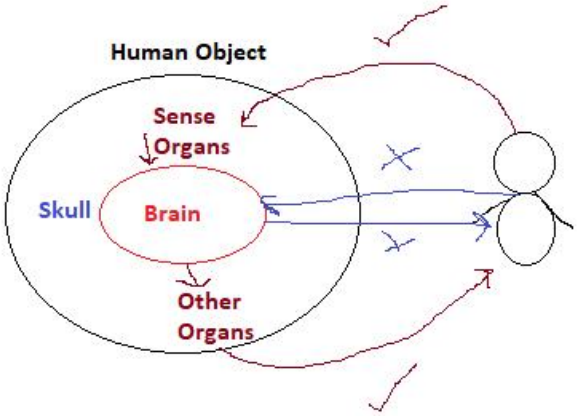
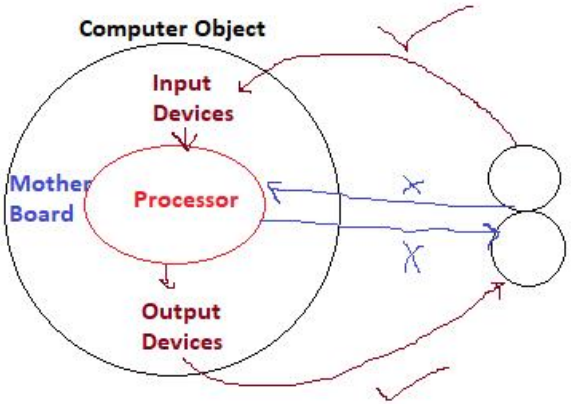

OBJECT ORIENTATION
ENCAPSULATION
INHERITANCE
POLYMORPHISM
ABSTRACTION

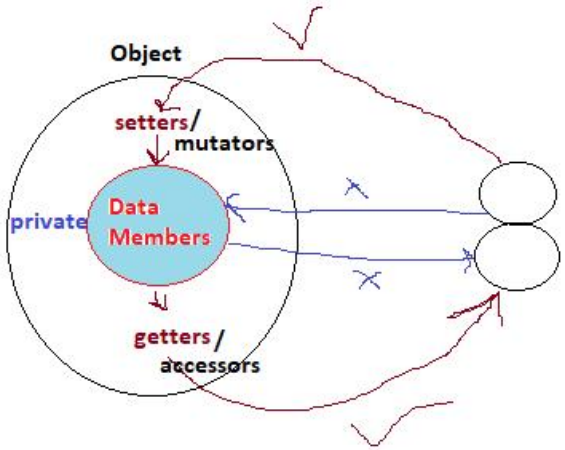
Data Encapsulation in Java

In Real World,



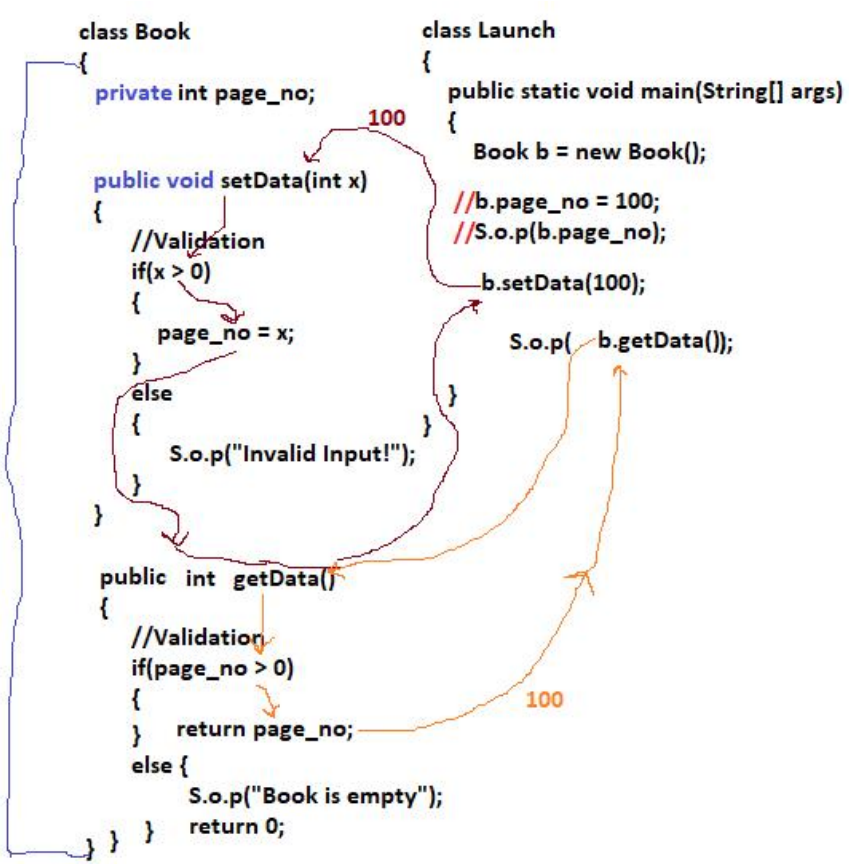


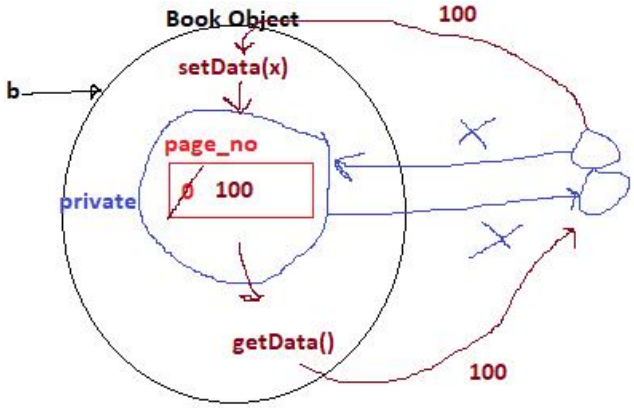
In Java World,



private --> access modifier
setters & getters --> public methods

Encapsulation refers to the process of providing security to the most important components of an object.
In Java, the most important components of an object are the data members.
Security to the data members can be provided by preventing direct access & providing controlled access.
Direct access can be prevented by declaring the data members as 'private'.
Controlled access can be provided by using 'public setters & getters' or 'accessors & mutators'.


class Book
{
private int page_no;
public void setData(int x)
{
//Validation
if(x > 0)
{
page_no = x;
}
else
{
S.o.p("Invalid Input!");
}
}
public int getData()
{
//Validation
if(page_no > 0)
{
return page_no;
}
else {
S.o.p("Book is empty");
return 0;
} }
} }
class Launch
{
public static void main(String[] args)
{
Book b = new Book();
//b.page_no = 100;
//S.o.p(b.page_no);
b.setData(100);
S.o.p(b.getData());
}
}
100
100


Book Object
100
setData(x)
b
page_no
private
100
getData()
100

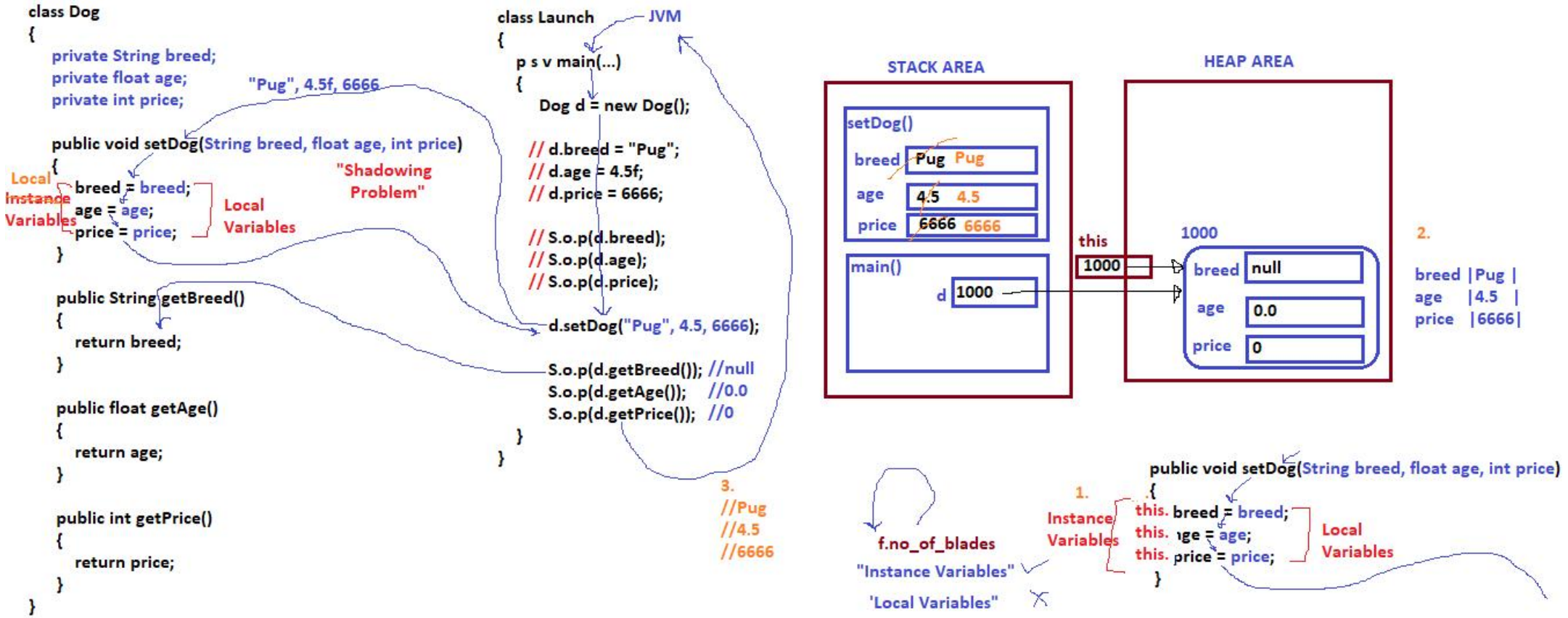
```
class Dog
{
    private String breed;
    private float age;
    private int price;

    public void setBreed(String x)
    {
        //Validation
        breed = x;
    }

    public void setAge(float y)
    {
        age = y;
    }

    public void setPrice(int z)
    {
        price = z;
    }
```

```
    public String getBreed()
    {
       //Validation
       return breed;
    }

    public float getAge()
    {
        return age;
    }

    public int getPrice()
    {
        return price;
    }
}
```

```
public void setDog(String x, float y, int z)
{
    breed = x;
    age = y;
    price = z;
}
```

```
public ??? getDog()
{
   return breed;
   return age;
   return price;
}
```

```
return_type    method_name(param1, param2, param3,....paramN)
{
        //body of the method

}
```


class Dog
{
private String breed;
private float age;
private int price;
public void setDog(String breed, float age, int price)
{
breed = breed;
age = age;
price = price;
}
public String getBreed()
{
return breed;
}
public float getAge()
{
return age;
}
public int getPrice()
{
return price;
}
}
"Pug", 4.5f, 6666
Local Instance Variables
Local Variables
"Shadowing Problem"
class Launch
{
p s v main(...)
{
Dog d = new Dog();
// d.breed = "Pug";
// d.age = 4.5f;
// d.price = 6666;
// S.o.p(d.breed);
// S.o.p(d.age);
// S.o.p(d.price);
d.setDog("Pug", 4.5, 6666);
S.o.p(d.getBreed()); //null
S.o.p(d.getAge()); //0.0
S.o.p(d.getPrice()); //0
}
}
JVM
3.
//Pug
//4.5
//6666
STACK AREA
HEAP AREA
setDog()
breed Pug Pug
age 4.5 4.5
price 6666 6666
main()
d 1000
this
1000
1000
breed null
age 0.0
price 0
2.
breed |Pug |
age |4.5 |
price |6666|
f.no_of_blades
"Instance Variables"
'Local Variables"
1.
Instance Variables
public void setDog(String breed, float age, int price)
{
this. breed = breed;
this. ıge = age;
this. price = price;
}
Local Variables

```
class Dog
{
    private String breed;
    private float age;
    private int price;

     public Dog(String breed, float age, int price)
     {
         this.breed = breed;
         this.age = age;
         this.price = price;
     }

     public String getBreed()
     {
        return breed;
     }

     public float getAge()
     {
        return age;
     }

     public int getPrice()
     {
        return price;
     }
}
```

```
class Launch
{
   p s v main(...)
   {
       Dog d = new Dog("Pug", 4.5f, 6666);
                                    constructor call
       S.o.p(d.getBreed());
       S.o.p(d.getAge());
       S.o.p(d.getPrice());
   }
}
```

Shadowing problem in Java:

It is a convention in Java that within a setter, the local variables should have the same name as that of the instance variables to improve code readability.
However, because of this convention, a name clash occurs between the local variables and the instance variables as a result of which local variables would shadow the instance variables. Hence, this name clash is referred to as "Shadowing Problem".

The shadowing problem can be resolved by using "this keyword".
this is a reference that refers to the current object - the object whose constructor or method is called.

Constructor in Java:

A constructor is a specialized setter which has the same name as that of the class.
A constructor does not return any value & hence it cannot have a return type.
A constructor is invoked during the object creation.
A constructor is used to initialize the newly created object.

```
class Dog   extends Dog
{
    private String breed
    private float age;
    private int price;

    Dog()
    {
       super();
    }


    public String getBreed()
    {
       return breed;
    }

    public float getAge()
    {
       return age;
    }

    public int getPrice()
    {
       return price;
    }
}
```

```
class Launch      "Default Constructor"
{
   p s v main(...)
   {
       Dog d = new Dog();
                                  constructor call
       S.o.p(d.getBreed());
       S.o.p(d.getAge());
       S.o.p(d.getPrice());
   }
}
```

```
class Dog
{
   private String breed;
   private float age;
   private int price;

   public Dog(String breed, float age, int price)
   {
      this.breed = breed;
      this.age = age;
      this.price = price;
   }

   public Dog()
   {
      breed = "BullDog";
      age = 5.5f;
      price = 7777;
   }

   public String getBreed()
   {
      return breed;
   }

   public float getAge()
   {
      return age;
   }

   public int getPrice()
   {
      return price;
   }
}
```

"CONSTRUCTOR OVERLOADING"

```
class Launch
{
  p s v main(...)
  {
  Dog d1 = new Dog("Pug", 4.5f, 6666);

  S.o.p(d1.getBreed()); //Pug
  S.o.p(d1.getAge());   //4.5
  S.o.p(d1.getPrice()); //6666

  Dog d2 = new Dog();

  S.o.p(d2.getBreed()); //BullDog
  S.o.p(d2.getAge());   //5.5
  S.o.p(d2.getPrice()); //7777

  }
}
```