


The background is a dark, textured grey. It is decorated with numerous water droplets of various sizes. Some droplets are large and prominent, while others are small and scattered. They are primarily located in the top-left and bottom-right corners, with a few smaller ones in the center and along the edges. The droplets have a realistic, glossy appearance with highlights and shadows.

OBJECT ORIENTED PROGRAMMING

© VIVEK DUTTA MISHRA, 2018



KEY ELEMENTS OF OOP

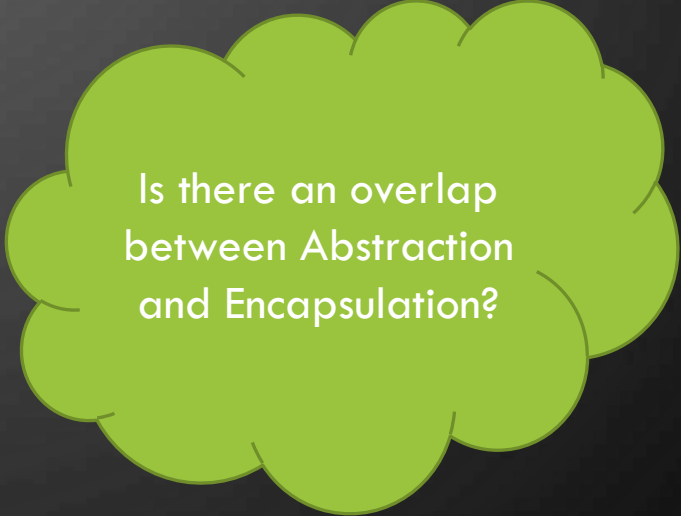
- OFTEN REFERRED AS 4 PILLARS OF OOP
 - ABSTRACTION
 - ENCAPSULATION
 - INHERITANCE
 - POLYMORPHISM
- 

ABSTRACTION (POPULAR VIEW)

- WHAT?
 - HIDING IMPLEMENTATION
 - CONCENTRATE ON WHAT AND NOT HOW
- HOW ?
 - SCOPE RULES
- WHY ?
 - ACCESS CONTROL

ENCAPSULATION (POPULAR VIEW)

- WHAT?
 - BINDING DATA AND METHODS TOGETHER
 - ACCESS DATA IN ORGANIZED WAY
- WHY?
 - DATA HIDING
- HOW?
 - ACCESS SPECIFIERS



Is there an overlap
between Abstraction
and Encapsulation?

INHERITANCE (POPULAR VIEW)

- WHAT?
 - PARENT-CHILD RELATIONSHIP
 - SPECIALIZATION
- WHY?
 - CODE REUSE

POLYMORPHISM (POPULAR VIEW)

- WHAT?
 - ONE NAME MANY FORMS
 - ABILITY TO TAKE MANY FORM
 - WHOSE ABILITY???
- WHY?
 - PERFORM DIFFERENT ACTION BASED ON SAME NAME
 - OBJECT BEHAVE DIFFERENTLY ON DIFFERENT CONTEXT
- WHY?
 - OVERLOADING
 - OVERRIDING

HOW DOES THIS DEFINITION HELP US

- HOW DOES THESE FEATURES HELP US?
- DATA HIDING IS ABTRACTION OR ENCAPSULATION?
- WHAT IS THE BENEFIT OF POLYMORPHISM?
- WHEN TO APPLY THESE FEATURES?
- WHAT IS THE BEST PRACTICE GUIDELINES?

BUDDHA ENLIGHTENMENT

When Siddharth became enlightened Buddha, learned men from around the world asked him what have you learnt?

2000 years later Sun Microsystems decided to do a C++ --



Buddha Replied – I have learnt nothing ... but un-learnt a lot of un-needed things I had gathered along the way in my life

Today it is our goal.

CASE STUDY : ANIMAL OBJECTS

- Which of the two approach below best describes Object 'snake' and 'tiger' why?

//Approach A

```
Animal tiger=new Animal(AnimalType.Tiger);  
Animal snake=new Animal(AnimalType.Snake);
```

//Approach B

```
Tiger tiger=new Tiger();  
Snake snake=new Snake();
```

ANIMALS: APPROACH A

```
//Approach A  
Animal tiger=new Animal(AnimalType.Tiger);  
Animal snake=new Animal(AnimalType.Snake);
```

- Design Consequences
- All properties of Tiger and Snake (and any other Animal) is defined in Animal class
- Since Snake has poison
 - Animal class defines poison
 - Since Animal class has poison
 - Tiger Has Poison.
- Since Crocodile LaysEgg
 - Animal class defines LayEgg
 - Tiger LayEgg

Here class
defines Object
or Object
defines class?



ANIMALS : APPROACH B

```
//Approach B  
Tiger tiger=new Tiger();  
Snake snake=new Snake();
```

- EACH OBJECT IS DEFINED BY ITS OWN CLASS
- EACH OBJECT CONTAINS EXACT DEFINITION OF THE OBJECT
- ITS AN OBJECT ORIENTED DESIGN
- BUT WHAT CONNECTS TIGER AND SNAKE TOGETHER?

NEW CASE STUDY

- **Phase 1 – Define the Crow Class**

```
class Crow{  
  
    public virtual Color GetColor(){  
        return Color.black;  
    }  
    public Egg LayEgg(){...}  
    public void Fly() {...}  
}
```

PHASE 2 – DEFINE PARROT CLASS

```
class Crow{  
  
    public virtual Color GetColor() {  
        return Color.black;  
    }  
    public Egg LayEgg() {...}  
    public void Fly() {...}  
}
```

```
class Parrot : Crow{  
  
    public virtual Color GetColor() {  
        return Color.green;  
    }  
    //Inherits & Reuse Fly() and LayEgg()  
  
}
```

```
[Test]  
public void CrowsAreBlack(){  
    Crow crow=new Parrot();  
    Assert.AreEqual( Color.Black, crow.GetColor());  
}
```

Assertion Failed: Expected Black Found Green

PHASE 2 – DEFINE PARROT CLASS

```
class Crow{
```

```
    public virtual Color GetColor() {  
        return Color.black;  
    }
```

```
    public Egg LayEgg() {...}  
    public void Fly() {...}
```

```
}
```

```
class Parrot : Crow{
```

```
    public virtual Color GetColor() {  
        return Color.green;  
    }
```

```
    //Inherits & Reuse Fly() and LayEgg()
```

```
}
```

[Test]

```
public void ParrotBabiesAreParrot(){  
    Parrot parrot=new Parrot();  
    Object baby= parrot  
                .LayEgg()  
                .Hatch();
```

```
    Assert.InstanceOf<Parrot>(baby);
```

```
}
```

Assertion Failed: Expected Parrot Found Crow

WHAT WENT WRONG?

- DESIGN WAS CREATED WITH THE THE MOTIVATION
 - INHERITANCE IS FOR CODE RE-USABILITY
- BUT DESIGN TURNED OUT TO BE WRONG
- WHATS THE CONCLUSION?
 - INHERITANCE MAY NOT BE FOR REUSABILITY

PARENT CHILD RELATION

- Which of the two better represents Parent-Child relation???

//Approach A

```
Father rdm=new Father("RDM");  
Son son=new Son("VDM");
```

//Approach B

```
Person rdm=new Person("RDM");  
Person son=new Person("VDM");
```


PARENT CHILD RELATION – APPROACH A

```
//Approach A  
Father rdm=new Father("RDM");  
Son son=new Son("VDM");
```

- “rdm” was born Father and was never Son
- “rdm” is universal (and absolute) father and nothing else
- QUIZ
 - if rdm has 3 kids and 3 lack in bank balance how much each kid gets to inherit (assume no biase)?

PARENT CHILD RELATION – APPROACH B

```
//Approach B
```

```
Person father=new Person("RDM");
```

```
Person son=new Person("VDM");
```

- FATHER AND SON ARE TWO OBJECTS OF THE SAME CLASS
- HOW WILL SON INHERIT FATHER'S BANK ACCOUNT?

```
//son inherit fathers bank balance
```

```
son
```

```
    .bankAccount
```

```
    .deposit(
```

```
        father
```

```
            .bankAccount
```

```
            .withdraw(amount)
```

```
    );
```

TAKEAWAY

- INHERITANCE IN REAL WORD IS AN OBJECT TO OBJECT RELATIONSHIP (OFTEN OF SAME TYPE)
- INHERITANCE IN PROGRAMMING IS CLASS TO CLASS RELATIONSHIP
 - IT DOESN'T REPRESENT PARENT CHILD RELATIONSHIP
 - IT REPRESENTS : CLASSIFICATION AND SUB-CLASSIFICATION
- INHERIT ONLY IF YOU FIND A RELATIONSHIP THAT CAN BE EXPRESSED AS “IS A TYPE OF”
- DO NOT INHERIT FOR
 - REUSE
 - HAS A
 - IS LIKE A
 - IS SIMILAR TO
 - ANY OTHER REASON.

ANOTHER BAD EXAMPLE

```
class HardDisk{  
  
    public int GetSpeed(){  
        //read-write 3 sets of data  
        //calculate average  
        return mbps;  
    }  
}
```

What defines
the speed of a
computer?

```
class Computer : HardDisk{  
  
    //No code  
}  
  
void main(){  
  
    Computer c=new Computer();  
    print( c.GetSpeed());  
    //what will be the output  
}
```

TAKEAWAY

- WHEN YOU INHERIT
 - YOU NOT ONLY GET TO REUSE IMPLEMENTATION
 - YOU ALSO GET THE TO BEHAVE (COPY INTERFACE) AS BASE OBJECT
- INHERITANCE REPRESENTS “IS A TYPE OF RELATIONSHIP”
- INHERITANCE IS ABOUT BECOMING
- THERE IS A DIFFERENCE ‘USING’ AND ‘BECOMING’

WHAT IS THE SOLUTION?

```
class HardDisk{

    public int GetSpeed(){
        //read-write 3 sets of data
        //calculate average
        return mbps;
    }
}

class Computer :-HardDisk{
    HardDisk hdd;
    CPU cpu;

    public double GetSpeed(){return cpu.GetSpeed(); }
    public int GetHddSpeed() { return hdd.GetSpeed(); }
}
```

Is it Reuse?
Remember we are
wrting new code
here?

A wrapper code that
actually reuses actual
logic

WHAT HAVE WE LEARNT?

- WE HAVE MANAGED TO REUSE WITHOUT INHERITANCE
- WHAT IS THIS MECHANISM CALLED?
 - HAS A
 - CONTAINS
 - **ENCAPSULATION**



WHAT IS ENCAPSULATION ALL ABOUT?

- ENCAPSULATION IS ABOUT DEFINING RESPONSIBILITY
 - ENCAPSULATION BINDS RESPONSIBILITY (BEHAVIOUR) WITH ITS STRUCTURAL REQUIREMENT
 - ENCAPSULATION ALLOWS TO REUSE THE OBJECT DEFINED WITH SPECIFIC RESPONSIBILITY.
- 

LAW #1 OF OO DESIGN

- PREFER **HAS A (ENCAPSULATION)** OVER IS A (INHERITANCE)
- WHY?
 - HAS A IS DYNAMIC, SCALABLE AND REUSABLE RELATIONSHIP
 - HAS POTENTIAL TO BE OBJECT TO OBJECT RELATIONSHIP
 - IS A IS STATIC, NON-SCALABLE CLASS TO CLASS RELATIONSHIP
- REMEMBER
 - YOU CAN CHANGE WHATEVER YOU HAVE, YOU CAN CHANGE WHAT YOU ARE
 - YOU CAN HAVE MANY OF WHAT HAVE, YOU WILL ALWAYS BE ONE OF WHAT YOU ARE

“HAS A” VS “IS A”

- IF YOU HAVE A CELL PHONE, YOU CAN DECIDE NOT TO HAVE
 - YOU CANT CHOOSE TO BE NOT HUMAN
- IF YOU HAVE ONE CELL PHONE YOU CAN HAVE TWO
 - YOU CANT BE TWO TIMES HUMAN

LAW #1 OF OO EXPLAINED

- TRY TO CONVERT A IS RELATIONSHIP TO HAS A RELATIONSHIP
- VIVEK IS AN EMPLOYEE → VIVEK HAS AN EMPLOYEE
 - BOTH ARE NOT SAME.
 - OFTEN WE NEED TO CHANGE OUR DESIGN
- VIVEK HAS AN EMPLOYEMENT
 - IF YOU HAVE AN EMPLOYMENT YOU MAY CHOOSE TO
 - SET IT TO NULL
 - SET IT TO A DIFFERENT
 - HAVE TWO EMPLOYEMENTS



THINK!!!

- VIVEK IS NOT A TRAINER
 - VIVEK HAS THE ROLE OF A TRAINER
- ANAND IS NOT A DOCTOR
 - ANAND HAS THE PROFESSION OF A DOCTOR
- RDM IS NOT A FATHER
 - HE HAS THE RELATION OF BEING FATHER TO VDM

SO WHO IS VIVEK?

- “IS A” SHOULD BE SOMETHING WHICH SHALL NEVER CHANGE
- CAN WE SAY “VIVEK IS A HUMAN”
- HAVEN'T WE SEEN HUMAN TURNING INTO A DEMON?
- VIVEK IS NOT A HUMAN
 - VIVEK HAS A HUMAN BEHAVIOUR

TAKEAWAY

- **WE OFTEN MISTAKE A “HAS A” RELATIONSHIP FOR A “IS A” RELATIONSHIP**
- **EVEN THE MOST FUNDAMENTAL “IS A” RELATIONSHIP CAN BE TRANSFORMED TO “HAS A” RELATIONSHIP**
- **DOES THAT MEAN THERE IS NO USE CASE OF “IS A RELATIONSHIP” ?**
 - **THERE IS CERTAINLY A USE CASE OF “IS A RELATIONSHIP”**
 - **BUT WE ARE NOT SURE WHICH ONE IS TRUE IS A RELATIONSHIP**
 - **TRY CONVERTING ALL ISA TO HAS A**
 - **WHEN YOU FAIL, YOU GET THE TRUE RELATIONSHIP.**



RECTANGLE SQUARE PROBLEM

- TODO: CODE WALKTHROUGH
- 

BACK TO CROW-PARROT EXAMPLE?

- WHAT RELATES CROW AND PARROT?
- BUT WHAT IS A BIRD?
 - HOW DOES IT LOOK LIKE?
 - HOW DOES IT FLY?
 - HOW DOES IT BUILD ITS NET?
- IS THERE ANY CONCRETE BEHAVIOUR OF A BIRD?



WHAT MAKES BIRD AN IDEAL BASE CLASS?

- BIRD IS ABSTRACT
 - THERE IS NOTHING THAT IS JUST BIRD
- THE ONLY REASON BIRD CLASS EXISTS IS TO DEFINE THE CLASS HIERARCHY
- NOTHING TO REUSE

LAW #2 OF OO DESIGN

- PREFER ABSTRACT INHERITANCE OVER CONCRETE INHERITANCE
- YOUR BASE CLASS SHOULD BE ABSTRACT
- PROVIDES REQUIRED CLASS HIERARCHY
- NO SCOPE OF REUSE
- THERE IS NO REAL “IS A TYPE OF” RELATIONSHIP BETWEEN TWO CONCRETE CLASS
- “IS LIKE A” IS OFTEN MISINTERPRETED AS “IS A TYPE OF”



POLYMORPHISM REVISITED

- WHAT IS POLYMORPHISM?
 - DIFFERENT OBJECTS, BEHAVING DIFFERENTLY, WITHIN THE SAME CONTEXT
 - WHY POLYMORPHISM MAKES SENSE?
- 

WHAT MAKES CAR A GREAT DESIGN?



what if the car's tyre is not replaceable and the tyre bursts?

Remember : if a smaller component of a system is not independently replaceable one day the system would be throw away.

Replaceable Tyres



WHAT IS A SOFTWARE COMPONENT?

- A SOFTWARE COMPONENT IS A PIECE OF CODE THAT
 1. HAS A WELL DEFINED RESPONSIBILITY (INDEPENDENT)
 2. IS REUSABLE
 3. IS REPLACEABLE (FUTURE CHANGES)

HOW TO REPLACE A COMPONENT?

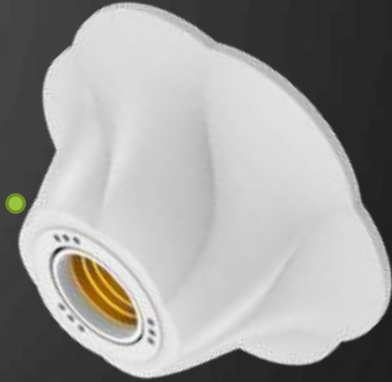
- YOU NEED A SOCKET
- WHERE COMPONENTS CAN BE PLUGGED.
- AND REPLACED
- IN SOFTWARE
 - A REFERENCE IS A SOCKET
 - AN OBJECT IS THE COMPONENT THAT IS PLUGGED TO IT



COMPARE SOCKETS

- WHICH OF THESE TWO SOCKET GIVES BETTER REPLACEABILITY?

Foolishly wise



- WHY?


- WHERE IGNORANCE IS A BLISS
IT IS FOOLISH TO BE WISE

Blissfully
Ignorant





POLYMORPHISM

- DIFFERENT OBJECTS BEHAVING DIFFERENTLY WITHIN THE SAME CONTEXT
 - CONTEXT IS POLYMORPHIC
 - CONTEXT IS LIKE A SOCKET
 - DIFFERENT OBJECTS CAN BE (POLYMORPHICALLY) PLUGGED INTO THE SOCKET
 - ALLOWS REPLACABILITY
- 

SAME OBJECT DIFFERENT BEHAVIOUR

```
class ParkerPen{  
  
    public void UseInHand(Hand hand){  
        print("writing");  
    }  
  
    public void UseInPocket(Pocket p){  
        print("status");  
    }  
}
```

- Is This Polymorphism?
- Why?

SAME OBJECT DIFFERENT BEHAVIOUR

```
class ParkerPen{  
  
    public void UseInHand(Hand hand){  
        print("writing");  
    }  
  
    public void UseInPocket(Hand hand){  
        print("status");  
    }  
}
```

Either both is
Polymorphism
or Neither is

```
class ParkerPen{  
  
    public void Use (Hand hand){  
        print("writing");  
    }  
  
    public void Use (Pocket p){  
        print("status");  
    }  
}
```

USING PARKER PEN

```
class ParkerPen{  
  
    public void Use (Hand hand){  
        print("writing");  
    }  
  
    public void Use (Pocket p){  
        print("status");  
    }  
}
```

```
void main(){  
    ParkerPen pen=new ParkerPen();  
  
    Object context= GetHandOrPocket();  
    //assume you got Hand  
  
    pen.Use(context);  
}
```

Not a
polymorphic
socket

which
behaviour will
be called

USING PARKER PEN

```
class ParkerPen{  
  
    public void Use (Hand hand){  
        print("writing");  
    }  
  
    public void Use (Pocket p){  
        print("status");  
    }  
}
```

You need
different socket
for different
objects

```
void main(){  
    ParkerPen pen=new ParkerPen();  
  
    Object context= GetHandOrPocket();  
    //assume you got Hand  
  
    pen.Use(context);  
  
    if (context is Hand)  
        pen.Use( (Hand) context);  
    else  
        pen.Use((Pocket) context);  
}
```

TAKEAWAY

- POLYMORPHISM PROVIDES A REPLACEABLE SOCKET FOR COMPONENTS
 - WE NEED THE ABILITY TO USE DIFFERENT OBJECT WITHIN THE SAME CONTEXT
- OVERLOADING DOESN'T GIVE THE SAME BENEFIT
 - WE HAVE A SINGLE OBJECT (NO REPLACEMENT)
- OVERLOADING CAN BE TERMED AS PSEUDO POLYMORPHISM

ABSTRACTION

- ABSTRACTION IS A PROCESS (WHAT)
- TO CREATE SIMPLIFIED GENERALIZED DESIGN (WHY)
- BY REDUCING THE IMPLEMENTATION DETAILS (HOW)



ABSTRACTION BY EXCLUDING

- OBJECTS IN REAL WORLD IS COMPLEX
- WE DON'T NEED ALL ITS FEATURES TO IMPLEMENT OUR DOMAIN REQUIREMENT
- WHATEVER WE DON'T INCLUDE IN DOMAIN IS ESSENTIALLY ABSTRACTED

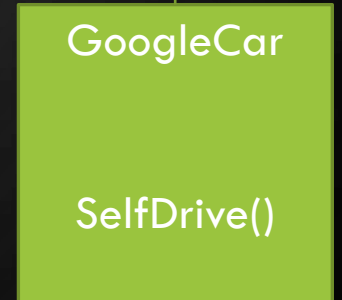
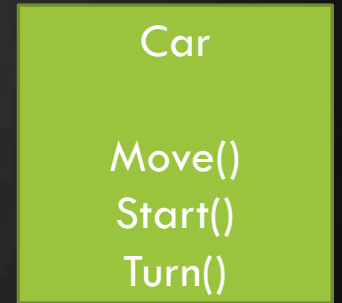
ABSTRACTION BY HIDING

- INFORMATION BROUGHT IN DOMAIN SHOULD BE USED RESPONSIBLY
- WE NEED TO PROTECT THE INFORMATION AND PROVIDE ACCESS TO PROPER OBJECT
- CAN BE DONE BY SCOPE RULES
- IMPLEMENTATION DETAILS HIDDEN USING PRIVATE
- ACCESS PERMITTED USING PUBLIC INTERFACE
- THE INTERFACE AND IMPLEMENTATION IS BONDED TOGETHER TO DEFINE THE RESPONSIBILITY



ABSTRACTION BY LAYERING (INHERITANCE)

- INFORMATION CAN BE LAYERED AS MULTIPLE ENCAPSULATED UNITS
- EACH LAYER EXTENDS THE OTHER
- EACH LAYER HAS REDUCED INFORMATION
- CLASSIFICATION AND SUB-CLASSIFICATION



Even
GoogleCar
Move and
Turn

ABSTRACTION BY SEPARATING INTERFACE AND IMPLEMENTATION

- SO FAR INTERFACE AND IMPLEMENTATION ARE BOUNDED AS A UNIT
- TO ALLOW IMPLANTATIONS SWAPPING (REPLACEMENT) WE NEED TO SEPARATE IT
- INTERFACE MATTER IMPLEMENTATION DOESN'T

Polymorphism

Transport

Move()



ABSTRACTION AND PIE

- ABTRACTION
 - GOAL OF A SOFTWARE
 - SIMPLIFIED, GENERALIZED DESIGN
 - BY REDUCING THE DETAILS
- ENCAPSULATION
 - DEFINES RESPONSIBILITY
 - BOUNDS BEHAVIOURS
 - UNIT OF REUSE

ABSTRACTION AND PIE

- INHERITANCE
 - DEFINES CLASSIFICATION AND SUB-CLASSIFICATION
 - CREATES A POLYMORPHIC HIERARCHY
 - BASE CLASS SHOULD BE ABSTRACT
 - ALLOWS OBJECTS TO BE TREATED AS SUB-TYPE OF ABSTRACTION
- POLYMORPHISM
 - ACHIEVED USING INHERITANCE HIERARCHY
 - DEFINES A SOCKET WHERE
 - INHERITED COMPONENTS CAN BE PLUGGED IN

Inheritance exists
to define
polymorphism.
The two aren't
different.

TAKEAWAY

- ABSTRACTION IS THE GOAL OF A SOFTWARE DESIGN SUPPORTED BY 3 KEY ELEMENTS – POLYMORPHISM , INHERITANCE AND ENCAPSULATION (PIE)
- ENCAPSULATION
 - DEFINES RESPONSIBILITY
 - DEFINES UNIT OF REUSE
- POLYMORPHISM
 - DEFINES SOCKET FOR REPLACEMENT
- INHERITANCE
 - TOOL TO ACHIEVE POLYMORPHISM

TAKEAWAY

- LAW #1 OF OBJECT ORIENTED DESIGN
 - PREFER “HAS A” OVER “IS A”
 - PROMOTES ENCAPSULATE TO REUSE
 - MANY “IS A” RELATIONSHIP IS ACTUALLY MISINTERPRETED “HAS A”
- LAW #2 OF OBJECT ORIENTED DESIGN
 - PREFER ABSTRACT INHERITANCE OVER CONCRETE INHERITANCE
 - CREATE POLYMORPHIC HIERARCHY
 - PROMOTES REPLACEABLE COMPONENT
 - MANY “IS A” RELATIONSHIP IS ACTUALLY MISINTERPRETER “IS LIKE A”

The image features a dark gray background with a subtle radial gradient. In the corners, there are several realistic-looking bubbles of varying sizes, some overlapping, creating a decorative frame. The bubbles have highlights and shadows, giving them a three-dimensional appearance.

WHAT HAVE WE LEARNT?