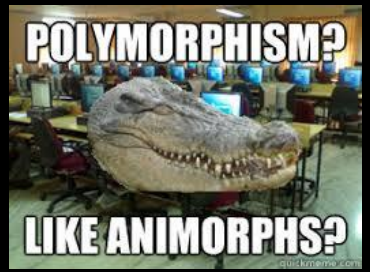
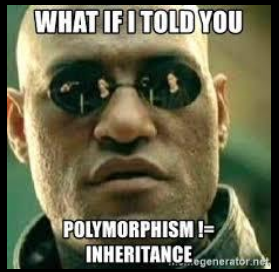
Polymorphism





Let's make it sound super complex tho

Learning about polymorphism for the first time.

It's object oriented!

Three Pillars of OO

- Encapsulation
- Inheritance (Specialization)
- Polymorphism (Abstraction)

Polymorphism

- Greek for “*having many forms*”
- Main Idea:
  - We can treat related objects as the *same kind* of object in client code.
  - At runtime, each object does the correct task

Example

```
class Pawn extends ChessPiece
{
	public Location[] getPossiblePawnMoves(Location origLocation, Gameboard board)
	{
		Location[] possibleMoves = new Location[3];
		possibleMoves[0] = new Location(myCurrentX, myCurrentY + 1);
		if (board.isOccupiedByBadGuys(new Location(myCurrentX + 1, myCurrentY + 1)
		{
			possibleMoves[1] = new Location(myCurrentX + 1, myCurrentY + 1);
		}
		//etc, etc, etc,
	}
}
```

Example -- note: Not valid code

```
class Rook extends ChessPiece
{
	public Location[] getPossibleRookMoves(Location origLocation, Gameboard board)
	{
		Location[] possibleMoves = new Location[10];
		for (int i = 0; i < 10; i++)
		{
			possibleMoves.add(new Location(myX + i; myY);
			possibleMoves.add(new Location(myX - i; myY);
			possibleMoves.add(new Location(myX; myY + i);
			possibleMoves.add(new Location(myX; myY - i);
		}
		return possibleMoves;
	}
}
```

Example - Client Code

```
for (ChessPiece piece : listOfChessPieces)
{
        Location[] possibleMoves;

        if (piece instanceof Pawn)
        {
                Pawn p = (Pawn) piece;
                possibleMoves = p.getPawnPossibleMoves(piece.getLocation, gameBoard);
        }
        else if (piece instanceof Rook)
        {
                Rook r = (Rook) piece;
                possibleMoves = r.getRookPossibleMoves(piece.getLocation, gameBoard);
        }
}
```

Question: What is wrong with this?

- Note: This code will work. It gets the job done
- Maintenance nightmare - imagine adding a new kind of piece
- If / Else statements are asking for errors
- Very verbose - lots of words
- Instead of telling objects to do stuff, we’re asking about them and then telling them *how* to do it.

How can we improve this?

Suggestions from the class

Step 1: Establish a common interface

```
abstract class ChessPiece
{
	public abstract Location[] getPossibleMoves(int cell);

		// Every concrete chess piece will have to implement this method!
}
```

Step 2: Implement that interface

Concept: Each subclass must be able to respond to a request to getPossibleMoves()

```
class Pawn extends ChessPiece
{
	@Override
	public Location[] getPossibleMoves()
	{
		return getPawnPossibleMoves();
		// Alternatively, you could just destroy getPawnMoves and write it here.
	}
}
```

```
class Rook extends ChessPiece {
	@Override
	public Location[] getPossibleMoves(Location origLocation, Gameboard board)
	{
		Location[] possibleMoves = new Location[10];
		for (int i = 0; i < 10; i++)
		{
			possibleMoves.add(new Location(myX + i; myY);
			possibleMoves.add(new Location(myX - 1; myY);
			possibleMoves.add(new Location(myX; myY + i);
			possibleMoves.add(new Location(myX; myY - i);
		}
		return possibleMoves;
	}
```

Step 3: Depend on Superclass in client

```
for (ChessPiece piece : listOfChessPieces)
{
        Location[] possibleMoves = piece.getPossibleMoves();
        // Do stuff with those moves.
}
```

Consequences of Using Polymorphism

1. Client code is shorter and cleaner
    a. Review: Client code is any code that *uses* our objects. Protip → All code is client code of *something*, so code accordingly.
2. BUT→ We won't know what the client code will do until runtime.
    a. Why? Because it depends on the actual runtime class of our object, which could be any subclass.

So, we trade one complexity for the other.

Any Questions?

Conceptual Question

Conceptual Question for you:

How does polymorphism affect extensibility?

Conceptual Answer

Polymorphism generally improves extensibility.

Why?

- Any new behavior we need to add can be added to subclasses, and our client code can remain unchanged

Abstraction

"All problems in computer science can be solved by another level of indirection, except of course for the problem of too many indirections."

-David Wheeler (1972-ish)

What is Abstraction?

Abstraction is considering *only* the relevant parts of an object and ignoring the others.

Example: Pawn

What we include:

Location l;
+ getPossiblePawnMoves
- getPossibleAttackMoves
- getPossibleMoveMoves

What we don’t include:

Shape s;
Weight w;
MagneticProperties mp;
MagicProperties mp2;
Person creator;
Philosphy personalBeliefSystem;

Consider a ChessPiece

Is a ChessPiece more or less abstract than Pawn?

→ More abstract. It has fewer details.

Example: ChessPiece

What we include:

Location l;

+ getPossibleMoves();

What we don’t include:

Errr, anything else.

Class Activity

We are modeling an Uber Taxi Simulator

- Will have cars, trucks, bikes
- Will have people and destinations
- These objects will move across some terrain (a grid, probably)
- Vehicles controlled by users
- Vehicles can pick humans up

Goal:

Spend four minutes talking to your neighbors about what classes exist in this simulation.

Sketch up (super quickly) some diagram of your choice that shows classes and their relationships

Class Activity 2

Now consider making a dinosaur like game

- Dinos move on a 2D Grid
- Can pick up and throw items at others
- Dinosaurs get bigger over time
- Dinosaurs move slower as they get hit
- Can pick up power-ups

Similarities Between Programs

- Moving on a 2D Grid
- Picking up Items
- User controlled objects

Dissimilarities

- One has dinosaurs
- etc.

Abstraction -->

Find the similarities that exist between the systems, and create a layer of abstraction that contains those similarities.

Delegate any specific differences to concrete subclasses

Abstraction -->

Note: The layer of abstraction that we create *cannot* stand on its own. That's why it is *abstract*.

So, what does this Abstract System do?

Polymorphism in Java

Objects

Generalization class: FarmAnimal

Specialization class: Chicken

```
ArrayList<FarmAnimal> farmAnimals = new ArrayList<>();
ArrayList<Chicken> chickens = new ArrayList<>();
chickens.add(pigmeoChicken);
farmAnimals.add(pigmeoChicken);


//farmAnimals.add(FarmAnimal farmAnimal)
//chickens.add(Chicken chicken)
FarmAnimal farmAnimal = new FarmAnimal();
Chicken chicken=new Chicken();
//farmAnimal = new Chicken();

farmAnimals.add(chicken);          //?1
farmAnimals.add(farmAnimal);       //?2

chickens.add(chicken);             //?3
chickens.add(farmAnimal);          //?4
```

Polymorphism in Java

Methods

Overriding: new definition of an existing method (inheritance)

Overloading: new definition to methods in the same class

```
@Override
public String toString(){
          //TODO
}

class FarmAnimal{
//...
   public float computeFoodCost() {
      //TODO To code or not to code
      return 0;
   }
}
```

```
class Chicken extends FarmAnimal{
  //...

  @Override
  public float computeFoodCost(){
    //TODO code
    //TODO code to compute cost to food

     return cost;
  }
}
```