Object-Oriented Programming

Session: Week 8 Session 1
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

Agenda:

1. Review this week's Assignments
2. Introduce the week's Learning Objectives
3. Review Learning Objectives
   - Threading
   - Database
   - Network Programming
4. Review Session 1 Topics

Our learning objectives from the syllabus assumed that we would not have focused quite so much on threading earlier in the session, so our threading discussion will be brief this week. Previous session of the course also focused somewhat more on database and SQL programming and a little less on network programming. With the continued expansion of Web Service and Cloud Computing in the industry, I am going to reverse that priority. We will still cover both topics; however, are focus will be Web Services and network programming while still recognizing the importance of databases.

Review Questions Assignment

Week 8 Questions Assignment [link]

Week 7 Questions Assignment:
http://www.epogue.info/CPSC-24500/Week07/2017SpringW07QuestionsAssignment.docx

Review Programming Assignment

Week 8 Programming Assignment [link]

Week 8 Programming Assignment:
http://www.epogue.info/CPSC-24500/Week08/2017SpringW08ProgrammingAssignment.pdf

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

- We covered these topics in week 5/6 and week 1 respectively.

Performance Optimization and Threading

Performance is critical in application development… the focus of performance optimization continues to evolve, but the criticality remains very high! Multithreading is one very important way that we can optimize CPU performance; however, there are many other performance bottlenecks and optimization techniques:

- CPU… threading
- Memory… optimize disk usage, buy more memory
- Disk IO… buffering, file size, or faster (more expensive) disks
- Network bandwidth… "file" or package size
- Network latency… pray for a miracle!
- User Interaction and Capabilities

I believe that performance optimization is one of the most important and challenging aspects of developing high quality software.

Consider mobile phone networks and satellite networks… and latency.

User experience.

Know the difference between latency and bandwidth and how it impacts network and application performance:
https://en.wikipedia.org/wiki/Network_performance

The following measures are often considered important:
Bandwidth commonly measured in bits/second is the maximum rate that information can be transferred
Latency the delay between the sender and the receiver decoding it, this is mainly a function of the signals travel time, and processing time at any nodes the information traverses
Throughput is the actual rate that information is transferred
Jitter variation in packet delay at the receiver of the information
Error rate the number of corrupted bits expressed as a percentage or fraction of the total sent

You can usually buy more bandwidth. Fixing a latency issue might require you to change the speed of light.

Threads & Multithreaded Applications

Multithreading: A technique by which a single set of code can be used by several processors or cores at different stages of execution.

- Threads and application performance are becoming nearly synonymous
- Moore's law only remains achievable if we can effectively utilize multi-processor, multi-core, and multi-threaded applications
- Our performance principles that we discuss will be applicable across platforms and environments
- Our practical focus will be on Java multi-threading
- Parallel processing has become the focus of the computing and software development industry
- The rise of big data, artificial intelligence, virtual/augmented reality, and dedicated graphical processing units (GPUs) have made that a nearly guaranteed trend for years to come

The difference between user perceived (real or perceived) is as important as actual performance. Optimizing application performance to reflect user capabilities is challenging and necessary. It doesn't matter if you optimize an application to be 10 times faster if the bottleneck is how fast the user.

Multithreading is the primary method of optimizing CPU performance. It is unlikely to be of benefit if the bottleneck is elsewhere. For example, if you are disk bound, splitting your process into multiple threads is unlike to be of benefit.

Processors, Cores, and Threads

- Computers have one or more Processors (CPUs)
- Processors each have one or more Cores
- Cores can create one or more Threads
- An application running only on one thread of a dual cpu, quad-core, single thread can utilize only a portion of $1/8^{th}$ of the processing power of that machine

For this discussion we will using multithreading and multiprocessing terms synonymously for our purposes.

Modern central processing units (CPUs) are made up of cores. A core is like a mini-processor that works with its fellow cores to perform the work that applications request of the CPU. In the old days, a CPU had just one core, a single channel through which all requests would pass. This was how we optimized applications… CPU, memory, fast disk, slow disk. Today, though, with multiple CPUs and multiple cores, a CPU can pay attention to and do many things at once.

This architecture, in turn, allows today's applications to perform multiple tasks at once. This ability is called multitasking. Multitasking enables an

Multi-Threaded Development

Now for the bad news. Multi-Threaded Development is really hard!

Image sources: Google

- Rigidity - It is hard to change because every change affects too many other parts of the system
- Fragility - When you make a change, unexpected parts of the system break
- Immobility - It is hard to reuse in another application because it cannot be disentangled from the current application

Stadia add-in net change example.

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order.

Multi-Threaded Development

Now for the bad news. Multi-Threaded Development is really hard!

- Developing commercial quality multi-threaded applications makes Rigidity, Fragility, and Immobility much harder to avoid
- Many of the 3rd party professional libraries that the industry had come to rely on came into question as multi-threading application became required
- Testing becomes harder when a sequence of events becomes variable
- What if your automated unit test results might be different depending on which thread finishes first?
- What about deadlock?
- **Performance** is so important that we will need to understand be able to effectively utilize, test, and deploy effective multi-threaded applications

- Rigidity - It is hard to change because every change affects too many other parts of the system
- Fragility - When you make a change, unexpected parts of the system break
- Immobility - It is hard to reuse in another application because it cannot be disentangled from the current application

Increased complexity is the primary disadvantage for developing multithreaded applications.

Some languages have come into existence in order to try to reduce the complexity of writing, enhancing, and supporting multithreaded applications. For example, Scala has implemented specific parallelization features in the core language that make it a first class threading language. Note that Scala also targets the Java runtime environment.

C++ would be an example of a language that has implemented a plethora of threading mechanisms for various platforms and implementations. Recent versions have introduced more common approaches.

Stadia add-in example.

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order.

Interesting threading article:

http://blog.smartbear.com/programming/why-johnny-cant-write-multithreaded-programs/

When reviewing libraries look for something like "This class is immutable and thread-safe." before you use it in multithreaded development.

FastPrime in C#

Write a performance optimized command line C# application that will programmatically find prime numbers and store those numbers sorted in an output file.

In FastPrime we will create a command line Java application that will:

1. Use multiple threads to find the prime numbers between two numbers
2. Sort those results and store them to a file
3. Perform some timings
4. … And do this all very fast

See the details in this week's assignment

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

We are reviewing this so that we can relate it back to databases and client-server development.

Object-Oriented Programming [link]

Object-oriented programming (OOP) is a programming model based on the concept of "objects", which contain both Attributes (data) and Methods (procedures) that operate on that those attributes.

Most popular OOP languages are class-based, meaning that objects are instances of classes.

It includes concepts, patterns, and principles for designing and implementing modern software products.

- Concepts – powerful features that prove indispensable to modern software development, brought to us automatically by object-oriented programming.
- Patterns – tried-and-true templates for forging relationships between classes
- Principles – guidelines that help you determine what classes are needed and how they should divide up the work

I will often start with definitions from Wikipedia and other sources. If you are struggling with a topic and/or would like more information, it can often be valuable to review the references. You should be able to click on the [link] tag (possibly while holding the shift key down) in order open the reference in a browser. Please let me know if this is not working for you.

Walking through the slide, you will see that words and terminology will be important as we discuss and learn new concepts. During the course I am sure you will notice that at times I will struggle with the attribute/property vs. data and method vs. procedure distinction when I am talking. I would like for us to try to make that distinction in our work as we go through the term.

For the purposes of this class, “attributes” and “properties” will be used interchangeably to describe variable belonging to a class or object. Also “procedures” and “functions” will be used interchangeably.

The “six” (Three plus) Object-Oriented Concepts

Object-oriented concepts:

1. **Encapsulation**… and Information Hiding
2. **Inheritance**… and Abstraction
3. **Polymorphism**

Plus… Composition & Aggregation

Helpful Interview Hint: Whenever you are asked a conceptual question about object-programming in an software development interview (and you will be), answer confidently “Encapsulation”, “Inheritance”, and “Polymorphism”.

When asked what is Encapsulation (or how would you implement it), say, “I would limit or minimize variable scope and keep data attributes private as often as possible.”

Now as we are going through our object-oriented examples, be thinking about how you would answer the “What is Inheritance?” and “What is Polymorphism?” interview questions. Note that answering them both with very brief examples can be very effective… and it is always best to use animals in you OOP interview examples.

Now we just need to make sure that we are able to effectively utilize these concepts after we get the job. Let’s start by walking through an example.

Encapsulation & Information Hiding

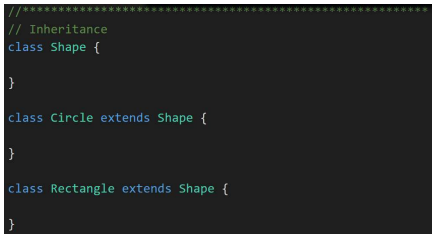
Encapsulation: Wrapping properties and methods into a class and minimizing the scope of those properties and methods.

Information Hiding: Minimize visibility/scope of data, attributes, functions, and methods.

Inheritance & Abstraction

Inheritance: When one class acquires the properties and methods of another class it is called Inheritance.

Abstraction: Something is abstract when it is a concept but is not concrete or defined enough to actually be built. Generally, in OO design, we start with abstract things, and then we build on them through Inheritance.



```
//*******************************************************
// Inheritance
class Shape {

}

class Circle extends Shape {

}

class Rectangle extends Shape {

}
```

The first of many “Shape” examples. We will get to a Abstraction example in a few minutes.

Polymorphism

Polymorphism: Polymorphism enables you to process collections of related things generically. This is particularly useful when you want to use a loop to march through a collection of items.

```
//*******************************************************
// Polymorphism
Shape[] shapes = new Shape[3];
shapes[0] = new Circle();
shapes[1] = new Rectangle();
shapes[2] = new Triangle();
for (Shape s : shapes) {
   System.out.println(s.area());
}
```

Example of polymorphism: a for loop that moves through the entries of a list. The list might be of a collection of related kinds of object. We can refer to each of the objects in the list through a generic variable (whose data type matches the one that all are ultimately related to). But, when we invoke a particular function that all members of the family share, each will respond by performing that function in their own specific way. For example, we could have a collection of Shape objects. We could refer to each entry in the Shape list through a generic Shape variable, even though the actual entries in the list are specific kinds of shapes – Circle, Rectangle, etc. All Shape objects might have the ability to calculate their own area. When we refer to an object in the list through a generic Shape variable and tell it to calculate its area, thanks to polymorphism, the circle version of the area() function will be called when we're dealing with a circle, and the Rectangle version of area() will be called when we're dealing with a rectangle, etc.

The first time through the for loop, we'll call the Circle.area() function – actually, we won't; it will happen automatically. The next time through, we'll call the Rectangle version, and then we'll call the Triangle version.

Polymorphism is implemented behind the scenes using a Virtual Method Table (VMT). The VMT keeps track of where various related classes' same-named functions are located in memory. Using the VMT, the operating system is able to figure out which code to implement when we tell each shape to fire its area() function.

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

We are reviewing this so that we can relate it back to databases and client-server development.

End of Session

Course Number: CPSC-24500

Week: 8

Session: 1

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 8 Discussion & Lecture (session 2)
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

Agenda:

1. Thank you!
2. Quickly Review this Week's:
   - To-do list
   - Programming Assignment
   - Questions Assignment
3. Review the week's Learning Objectives
4. Continue with More Learning Objective Topics
5. Programming Examples preview

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

We are reviewing this so that we can relate it back to databases and client-server development.

Data Bases [link]

A **database** is an organized collection of data. It is the collection of schemas, tables, queries, views, and other objects. The data are typically organized to model aspects of reality in a way that supports business processes requiring information.

A **database management system** (**DBMS**) is a application that interacts with the user, other applications, and the itself to capture and manage data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA, and IBM DB2.

- Schemas – defines that data constraints and definitions for tables and fields
- Tables – collection of related data held in a structured format that consists of columns and rows
- Views & Queries – used to create more easily accessible relationships without modifying the underlying physical schema

I will often start with definitions from Wikipedia and other sources. If you are struggling with a topic and/or would like more information, it can often be valuable to review the references. You should be able to click on the [link] tag (possibly while holding the shift key down) in order open the reference in a browser. Please let me know if this is not working for you.

Databases and Database Management Systems (DBMS) are a HUGE topic and one of the cornerstones of most IT organizations. We are just going to be touching the surface today.

We often inadvertently refer to database management systems as databases. We should try to be specific in our conversation. In addition, we generally use the term database and DBMS to infer a relational database.

Databases are generally not portable between DBMS systems. However, a level of portability can be obtained by utilizing industry standard tools and languages like SQL.

Spreadsheets are often used… inappropriately in many cases.

A database is not generally portable across different DBMSs, but different DBMS can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one DBMS. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language.

We generally attempt model our databases so that that they reflect real-world relationships. Optimal database design and optimal object-oriented design often diverge. When they do, you will need to make a

judgement call.

Normalization vs De-normalization:
https://en.wikipedia.org/wiki/Database_normalization

Database Normalization [link]

Database Normalization, or simply normalization, is the process of organizing the columns (attributes) and tables (relations) of a relational database to reduce data redundancy and improve data integrity.

Normalization is also the process of simplifying the design of a database so that it achieves the optimum structure. It reduces and eliminates redundant data. In normalization, data integrity is assured. It can also cause a performance tradeoff.

It may be important when you are doing object-modeling in your application to consider the database relationships and normalizing that will be important.

Database Normalization Example:

Consider the non-normalized and normalized table structure for a simple “checkbook” transaction database:

CustomerTransactionDatabase (non-nomalized):

CustomerTransactionTable:

FirstName	LastName	T1Date	T1Amount	T2Date	T2Amount	T3Date	T3Amount
Eric	Pogue	1/1/2017	-12.00	1/3/2017	-27.00	1/12/2017	-117.00
Ed	Smith	1/1/2017	-102.00	1/18/2017	-17.00	1/29/2017	-222.00

CustomerTransactionDatabase (nomalized):

CustomerTable:

FirstName	LastName	CustumerID
Eric	Pogue	1
Ed	Smith	2

TransactionTable:

CustomerID	Date	Amount
1	1/1/2017	-12.00
1	1/3/2017	-27.00
1	1/12/2017	-117.00
2	1/1/2017	-102.00
2	1/18/2017	-17.00
2	1/29/2017	-222.00

Note: Joins and/or Views can make the normalized tables look like the normalized table.

Phone Number Database Example

For example, a PhoneNumber database could be set up with two Tables. One table of Names and the other table of PhoneNumbers. For this simple example we might end up with something like:

- Schemas: Two tables one called Names and one called PhoneNumbers, text will be stored in UTF-8, we will allow duplicates, and set an index on Names.FirstName and Names.LastName.
- Names Table: Three fields including FirstName and LastName of UTF-8 text, maximum length 40 characters, and can not be NULL. The third field will be an autoincrement unique ID that is an integer and does not allow duplicates. All three fields would be indexed for fast searching.
- PhoneNumbers Table: Four fields including CountryCode, AreaCode, and Number which are all UTF-8 strings that can only contain numbers. Maximum lengths for each are 2, 3, and 7 respectively. The fourth field will be the unique ID of the person associated with the phone number. CountryCode can be NULL, other fields cannot. Fields will not be indexed as it will be rare that we have a phone number and want to lookup the person.
- Views & Queries – A query that Joins a set of Names to all of their respective phone numbers and shows that in a view would be valuable.

Structured Query Language (SQL) [link]

Structured Query Language or SQL is a domain-specific language used in programming and designed for managing data held in a relational database management systems. SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.

- Data Definition Language (DDL) – manages the table and index structure with common terms including CREATE, ALTER, RENAME, and DROP
- Queries – The most common operation in SQL makes use of the SELECT statement and generally includes a WHERE clause
- Many others…

I will often start with definitions from Wikipedia and other sources. If you are struggling with a topic and/or would like more information, it can often be valuable to review the references. You should be able to click on the [link] tag (possibly while holding the shift key down) in order open the reference in a browser. Please let me know if this is not working for you.

Databases and Database Management Systems (DBMS) are a HUGE topic and one of the cornerstones of most IT organizations. We are just going to be touching the surface today.

We often inadvertently refer to database management systems as databases. We should try to be specific in our conversation. In addition, we generally use the term database and DBMS to infer a relational database.

Databases are generally not portable between DBMS systems. However, a level of portability can be obtained by utilizing industry standard tools and languages like SQL.

Spreadsheets are often used

Student
Course
Semester
Room
Prerequisite Courses

A database is not generally portable across different DBMSs, but different DBMS can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one

DBMS. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language.

Relationships are based on the shaping of columns between col

Table definition

- guidelines that help you determine what classes are needed and how they should divide up the work
- such as modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.
- Natural relationship among different parts of the data

SQL Examples

```
SELECT *
FROM Names
WHERE LastName = Pogue
ORDER BY FirstName
```

```
DROP Names
```

```
SELECT isbn,
       title,
       price,
       price * 0.06 AS sales_tax
 FROM  Book
 WHERE price > 100.00
 ORDER BY title;
```

```
CREATE TABLE Names(
 column1 INTEGER Auto-Increment,
 column2 VARCHAR(50),
 column3 DATE NOT NULL,
 PRIMARY KEY (column1, column2)
);
```

```
SELECT b.isbn, b.title, b.price, sales.items_sold, sales.company_nm
FROM Book b
  JOIN (SELECT SUM(Items_Sold) Items_Sold, Company_Nm, ISBN
        FROM Book_Sales
        GROUP BY Company_Nm, ISBN) sales
  ON sales.isbn = b.isbn
```

Very powerful language for managing data and is often used as an “embedded” language from within other environments like Java, C#, and Python.

Data Base Management Systems Pro & Cons

Many of the most powerful DBMS strengths can also be their greatest weaknesses… particularly as it come to object-oriented development:

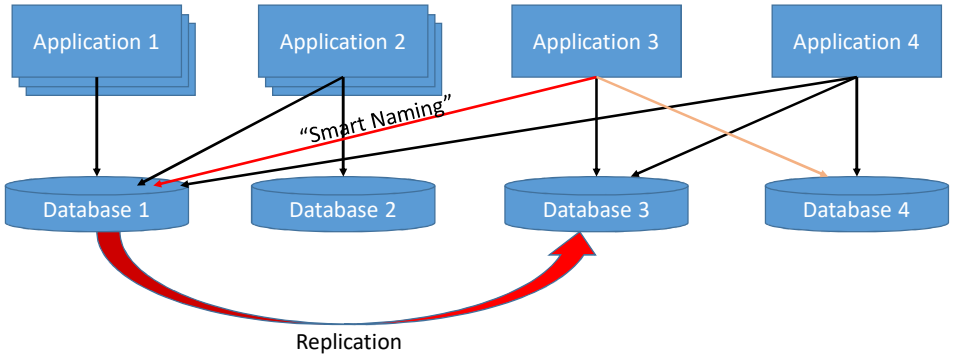
- Independent Data… vs Combining Functionality & Data
- Ubiquitous access… vs Encapsulation and Data Hiding
- Tables & Relationships… vs Inheritance
- Joining Tables to Create Relationships… vs Polymorphism
- Relationship Modeling… vs Object Modeling
- SQL… vs Java or C# or Python

Data Base Management Systems with OOP

Live in harmony. Embrace the database while keep a focus on OOP concepts:

- Store your data in a database
- Enforce all important relationships in the database
- Differentiate enterprise or shared data from application specific data… keep the shared data as small as is reasonable and have a separate person/team manage the shared data
- Don't utilize two-tier client-server architectures for anything more than small scale (less than 8 local users) application
- … And be very very careful about using tools that automate the connection between you two-teir view elements and their associated database elements
- Recombine you data, and functionality by creating three-tier application for more important applications
- Don't try to "automate" your object model to relationship model conversions
- Never, never, never uses "smart" or cute naming conventions to represent relationships!

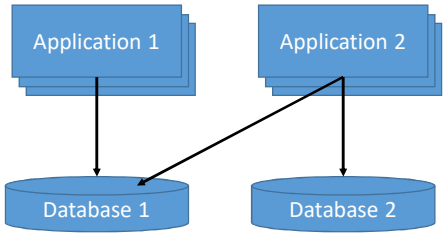
Two-tier and Three-tier Architectures



Replication is most often a bad idea.

"Smart Naming"

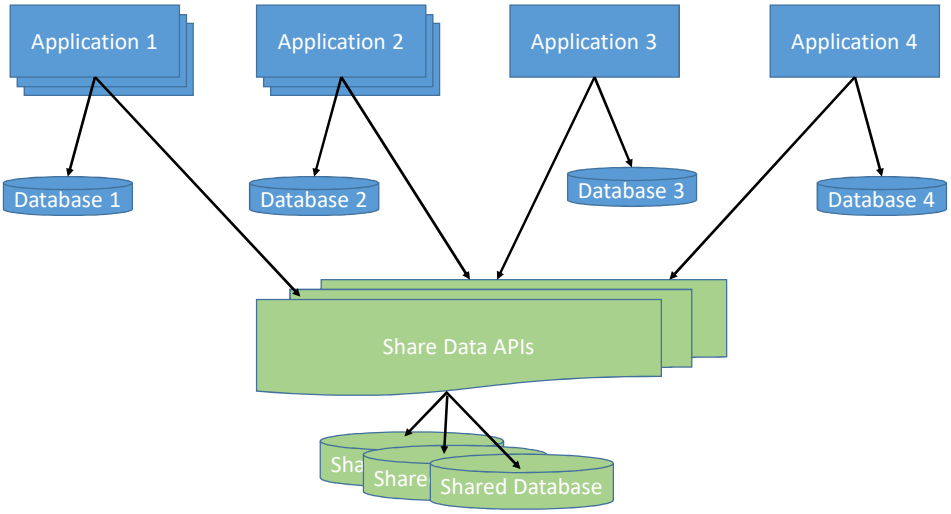
Two-tier and Three-tier Architectures



Replication is most often a bad idea.

“Smart Naming”

Two-tier and Three-tier Architectures

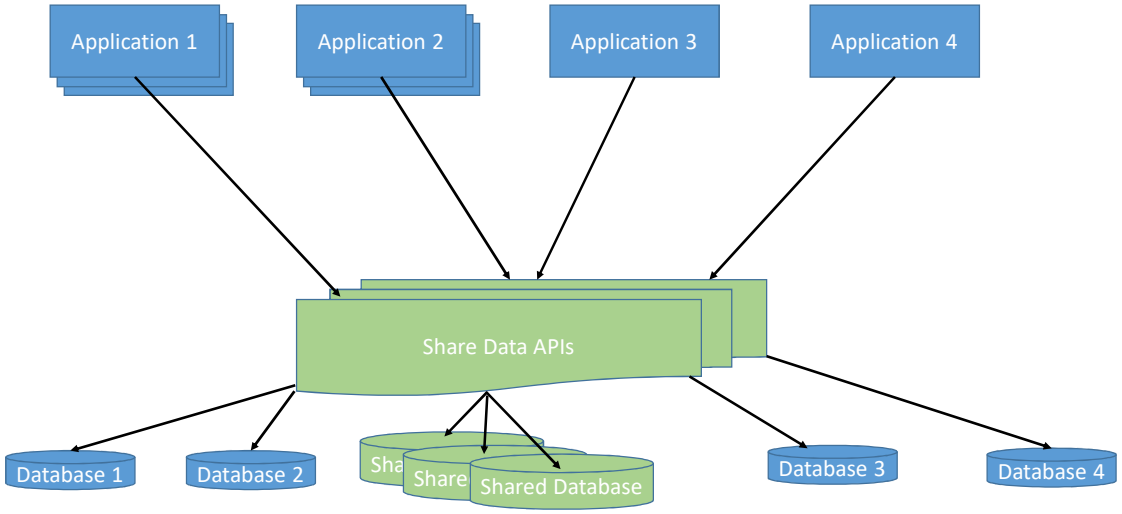


Cross-platform Enforced Encapsulation & Data Hiding!
Security
Sharing of business logic and processes.

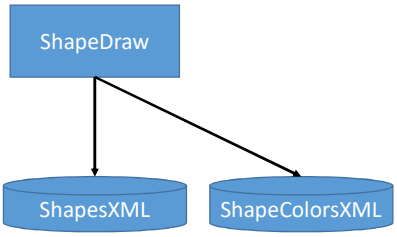
Interface Inheritance

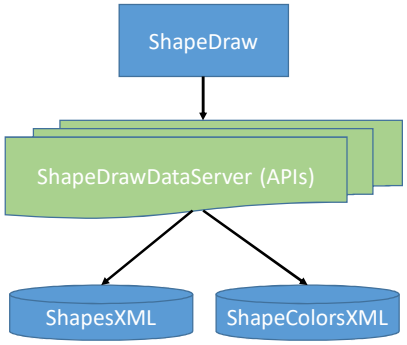
Very little benefit in Implementation Inheritance or Polymorphism

Two-tier and Three-tier Architectures



ShapeDrawDataServer Architecture





Replication is most often a bad idea.

“Smart Naming”

Review… Data Base Management Systems with OOP

Live in harmony. Embrace the database while keep a focus on OOP concepts:

- Store your data in a database
- Enforce all important relationships in the database
- Differentiate enterprise or shared data from application specific data… keep the shared data as small as is reasonable and have a separate person/team manage the shared data
- *Don't utilize two-tier client-server architectures for anything more than small scale (less than 8 local users) application*
- ***… And be very very careful about using tools that automate the connection between you two-teir view elements and their associated database elements*
- Recombine you data, and functionality by creating three-tier application for more important applications
- **Don't try to "automate" your object model to relationship model conversions
- ***Never, never, never uses "smart" or cute naming conventions to represent relationships!*

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

We are reviewing this so that we can relate it back to databases and client-server development.

Three-Tier Architecture Protocols & Formats

Three-Tier application architectures can use a variety of network protocols and formats including::

- TCP/IP: Transmission Control Protocol / Internet Protocol
- Sockets: Another term for TCP/IP
- HTTP: Hypertext Transfer Protocol
- HTTPs: Hypertext Transfer Protocol Secure
- SSL: Secure Sockets Layer
- XML or JSON: Extensible Markup Language
- SQL

Aggregated Protocol Standards:

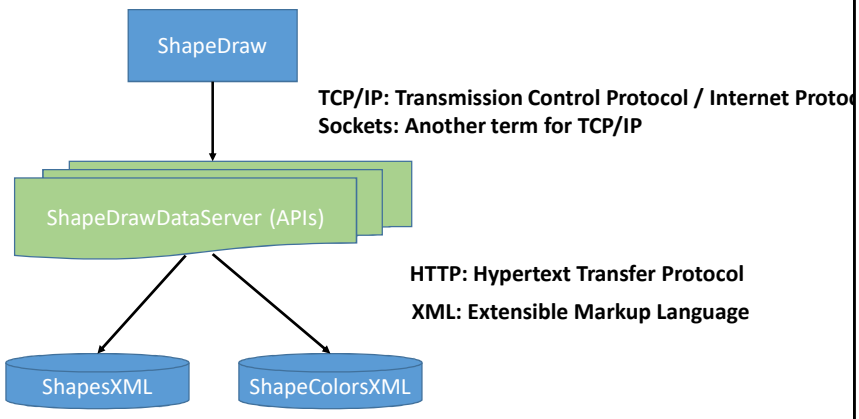
- SOAP: Simple Object Access Protocol (HTTP/HTTPs/Sockets with XML)
- REST: Representational State Transfer (HTTP/HTTPs with JSON)

ShapeDrawDataServer

Three-Tier application architectures can use a variety of network protocols and formats including::

- **TCP/IP: Transmission Control Protocol / Internet Protocol**
- **Sockets: Another term for TCP/IP**
- **HTTP: Hypertext Transfer Protocol**
- ~~HTTPs: Hypertext Transfer Protocol Secure~~
- ~~SSL: Secure Sockets Layer~~
- **XML: Extensible Markup Language**
- ~~SQL~~

ShapeDrawDataServer Architecture



Review ShapeDrawDataServer

Week 8 Programming Assignment [link]

Week 8 Programming Assignment:
http://www.epogue.info/CPSC-24500/Week08/2017SpringW08ProgrammingAssignment.pdf

Object-Oriented Programming

Session: Week 8 Session 3 Preview
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

ShapeDrawDataClient Application:

1. Develop application entirely in Visual Studio 2017 and C#
2. Take in one command line argument that is command to pass to ShapeDrawDataServer
3. Initiate a socket call to the server passing the command
4. Write out the response received from the sever
5. Wait for the user to press a key before shutting down… so that we can look at the response.

Object-Oriented Programming

Session: Week 8 Session 4 Preview
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

ShapeDrawDataServerStart Application:

1. Develop application entirely in Visual Studio 2017 and C#
2. Listen for a socket request
3. Respond with “Hello World!!!”
4. Provide starting point for ShapeDrawDataServer and “get-shapeandcolors;” request
5. You may utilize this code as the starting point for your Week 8 assignment

Starting point means that you should fully understand the code, but you can use it verbatim. You still must fix any errors or warning to get full credit.

End of Session

Course Number: CPSC-24500

Week: 8

Session: 2

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 8 Session 3
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

ShapeDrawDataClient Application:

1. Develop application entirely in Visual Studio 2017 and C#
2. Take in one command line argument that is command to pass to the server
3. Initiate a socket call to the server passing the command
4. Write out the response received from the sever to the console
5. Accept multiple requests as console input
6. Close the application when the user enters “end;” as console input

End of Session

Course Number: CPSC-24500

Week: 8

Session: 3

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 8 Session 4
Instructor: Eric Pogue

Agenda:

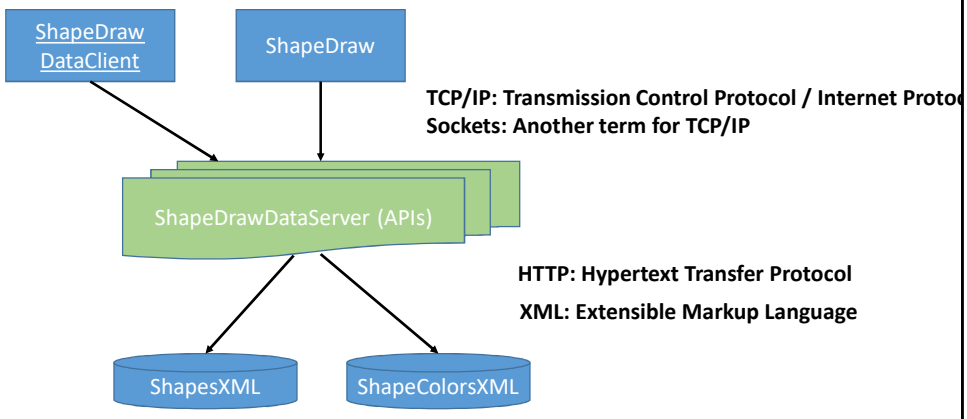
1. Review Week 8 Learning Objectives
2. Develop ShapeDrawDataServerStart Application
3. Review ShapeDrawDataClient Application
4. Test ShapeDrawDataClient and Server together
5. Next Steps

Learning Objectives – Week 8

1. Describe what a thread is and why it can be useful to distribute tasks among multiple threads
2. Review our multi-threaded application development activities
3. Explain why it is important to synchronize threads that need to share data source access
4. Review Object Oriented Programming benefits including the associating Data & Functionality, Encapsulation & Information Hiding, Inheritance, and Polymorphism
5. Review databases, database servers, and the SQL language
6. Understand how databases support (or don't support)work within a Object Oriented Programming environment
7. Understand client-server (two-tier), three-tier, and n-tier architectures
8. Introduce network programming concepts
9. Understand Web Services network programming
10. Develop a middle-tier data server using network programming

We are reviewing this so that we can relate it back to databases and client-server development.

ShapeDrawDataServer Architecture



Object-Oriented Programming

Session: Week 8 Session 4
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

ShapeDrawDataServerStart Application:

1. Develop application entirely in Visual Studio 2017 and C#
2. Listen for a socket request
3. Test Server and Client together
4. Reiterate that you may utilize this code as the starting point for your Week 8 assignment
5. Next Steps

End of Session

Course Number: CPSC-24500

Week: 7

Session: 4

Instructor: Eric Pogue

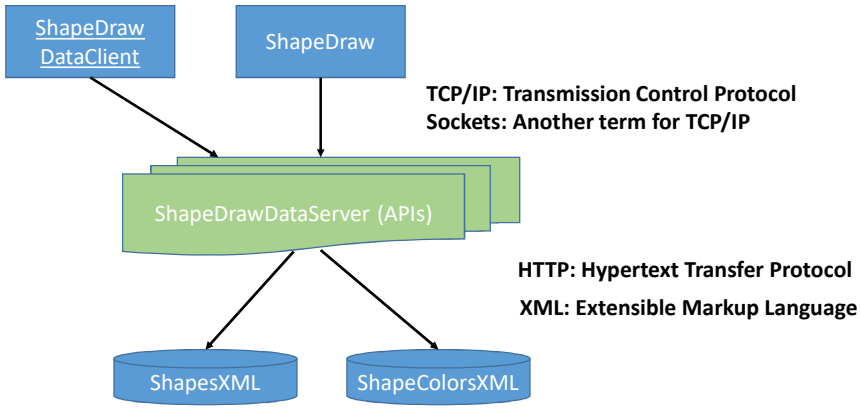
Object-Oriented Programming

Session: Week 8 Session 5
Instructor: Eric Pogue

ShapeDrawDataServerLite Application:

1. Develop application entirely in Visual Studio 2017 and C#
2. Download and compile ShapeDrawDataClient
3. Create a new ShapeDrawDataServerLite application
4. Copy and past ShapeDrawDataServerStart into “Lite”
5. Compile, run, and test “Lite”
6. Implement EJPShape and EJPShapeModel from Week 7
7. Download and parse InternetShapeDraw.xml
8. Create empty EJPColorModel class that always returns “blue”… you will need to download and parse the ShapeDrawColors.xml file as part of your assignment
9. Return a subset of the xml that will be needed for the assignment

ShapeDrawDataServer Architecture



End of Session

Course Number: CPSC-24500

Week: 8

Session: 5

Instructor: Eric Pogue