{OOP}

OBJECT ORIENTED CONCEPTS & DESIGN

AGENDA

- Introduction to Programming Languages
- Understand the 4 pillars of Object-Oriented Programming:

  Abstraction
  Inheritance
  Encapsulation
  Polymorphism
- C# Vs C++
- Game Design Patterns

PROGRAMMING LANGUAGES

PROGRAMMING LANGUAGES

- Programming languages allow programmers to code software
- The three major families of languages are:
  - Machine languages
  - Assembly language
  - High-Level languages

MACHINE LANGUAGES

- Comprised of 1s and 0s
- The native language of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail.
- Example of code:

```
1110100010101       111010101110
10111010110100      10100011110111
```

ASSEMBLY LANGUAGES

- Assembly languages are a step towards easier programming.
- Assembly languages are comprised of a set of elemental commands which are tied to a specific processor.
- Assembly language code needs to be translated to machine language before the computer processes it.
- Example:

```
ADD  1001010, 1011010
```

HIGH-LEVEL LANGUAGES

- High-level languages represent a giant leap towards easier programming.
- The syntax of HL languages is similar to English.
- Interpreted vs Compiled
- Historically, we divide HL languages into two groups:
  - Procedural languages
  - Object-Oriented languages (OOP)

PROCEDURAL LANGUAGES

- Early high-level languages are typically called procedural languages.
- Procedural languages are characterized by sequential sets of linear commands. The focus of such languages is on structure.
- Examples include C, COBOL, Fortran, LISP, Perl, HTML, VBScript

OBJECT-ORIENTED LANGUAGES

- Most object-oriented languages are high-level languages.
- The focus of OOP languages is not on structure, but on modelling data.
- Programmers code using "blueprints" of data models called classes.
- Examples of OOP languages include C++, C# and Java.

PROCEDURAL VS. OBJECT-ORIENTED PROGRAMMING

- The unit in procedural programming is ***function***, and unit in object-oriented programming is ***class***
- Procedural programming concentrates on creating functions, while object-oriented programming starts from isolating the classes, and then look for the methods inside them.
- Procedural programming separates the data of the program from the operations that manipulate the data, while object-oriented programming focus on both of them

OBJECT ORIENTED PROGRAMMING

CLASSES AND OBJECTS

CLASSES AND OBJECTS

- A **class** is a data type describing an object’s attributes (data) and methods (operations).
- An object is an instance of a class.
- This is analogous to a variable and a data type

OBJECT ORIENTED PROGRAMMING

- Attribute/Fields - Things which describe an object; the “adjectives” of objects.
- Methods - executable code of the class built from statements. It allows us to manipulate/change the state of an object or access the value of the data member - Verbs
- Constructors - Special methods used to create new instances of a class (Example: A Honda Civic is an instance of the automobile class.)

A SAMPLE CLASS

```
class Pencil {
    public String mColor = “red”;
    public int mLength;
    public float mDiameter;

    public static long sNextID = 0;

    public void SetColor (String inColor) {
            mColor = inColor;
        }
 }
```

FIELD MODIFIERS

- Private: private members are accessible only in the class itself
- Package/Namespace: package members are accessible in classes in the same package and the class itself
- Protected: protected members are accessible in classes in the same package, in subclasses of the class, and in the class itself
- Public: public members are accessible anywhere the class is accessible
- static: Only one copy

OBJECT ORIENTED CONCEPTS

OOP - ENCAPSULATION

- Incorporation of data & operations together in Class/Object
- Data can only be accessed through Class/Object
- “Information Hiding”

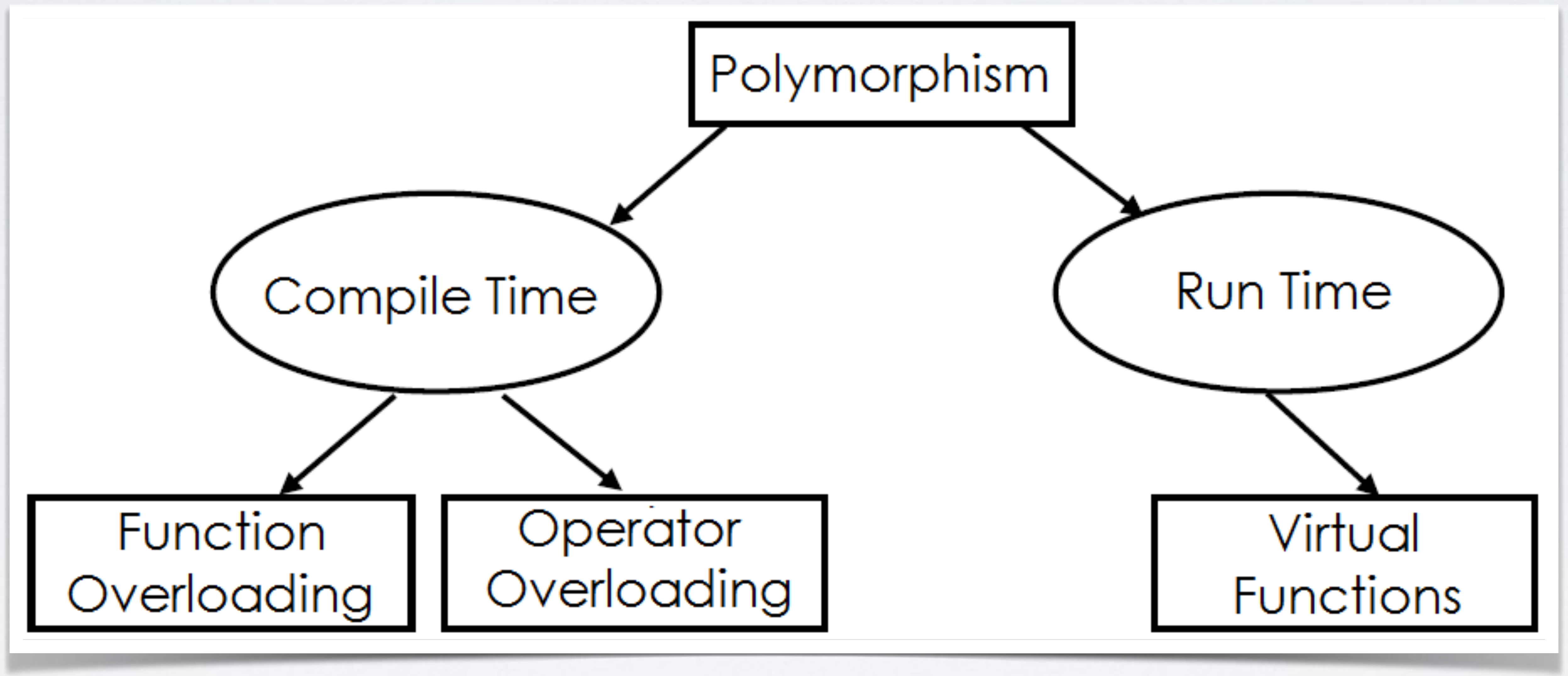
OOP - INHERITANCE

- Allows programmers to create new classes based on an existing class
- Methods and attributes from the parent class are inherited by the newly-created class
- New methods and attributes can be created in the new class, but don’t affect the parent class’s definition

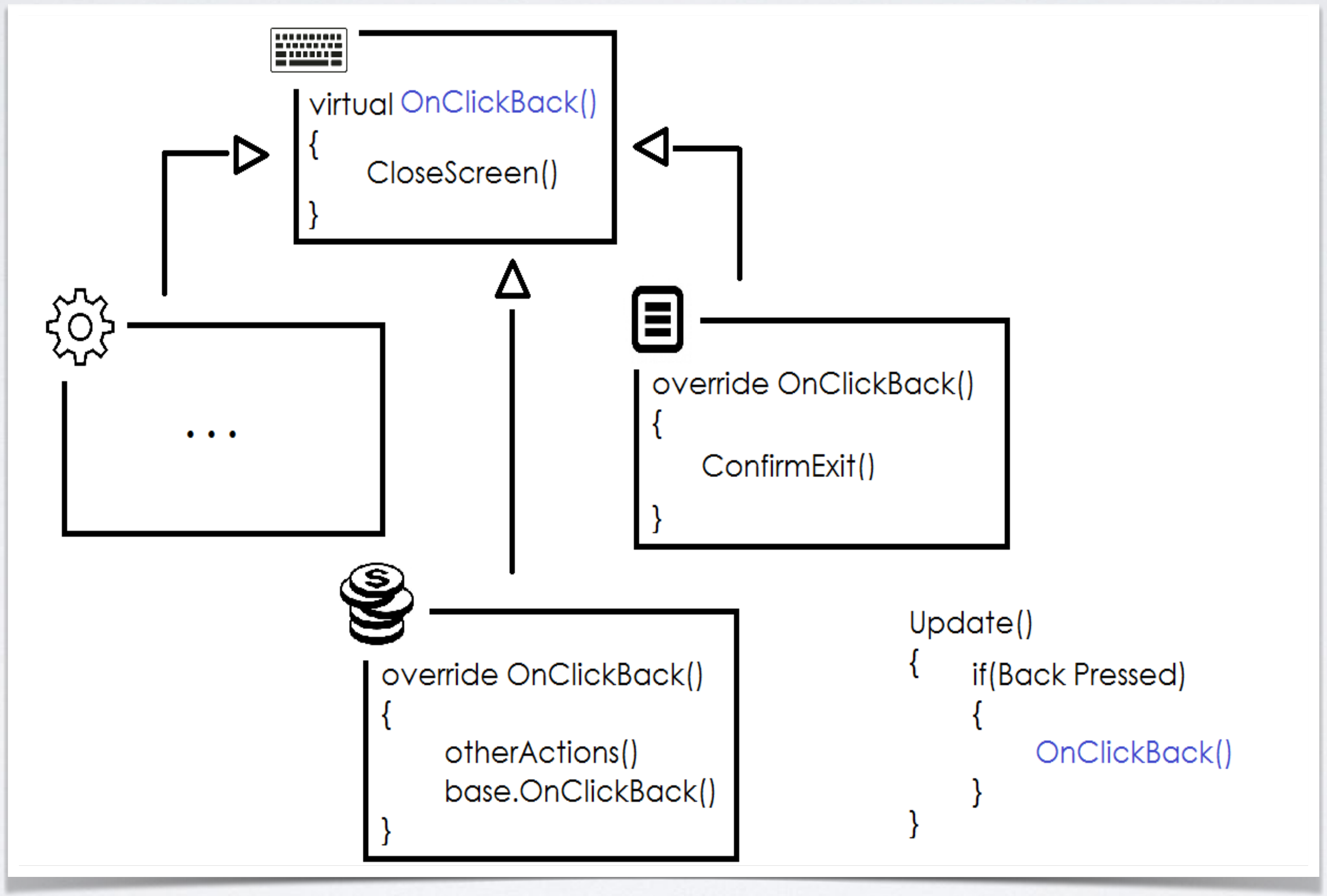
OOP - POLYMORPHISM

- Creating methods which describe the way to do some general function (Example: The “drive” method in the automobile class)
- Polymorphic methods can adapt to specific types of objects.

OOP - POLYMORPHISM



OOP - POLYMORPHISM



```
virtual OnClickBack()
{
    CloseScreen()
}
```

```
...
```

```
override OnClickBack()
{
    ConfirmExit()
}
```

```
override OnClickBack()
{
    otherActions()
    base.OnClickBack()
}
```

```
Update()
{
    if(Back Pressed)
    {
        OnClickBack()
    }
}
```

TEMPLATES

- Allows functions and classes to operate with generic typesPolymorphic methods can adapt to specific types of objects.

- allows a function or class to work on many different data types without being rewritten

```
template <typename T>
inline T max(T a, T b) {
    return a > b ? a : b;
}
```

C# VS C++

C# Vs C++

- No pointers required
- C# supports automatic garbage collection
- Unlike C++, C# supports foreach statement
- In C#, delegates, events and properties can also be specified as class members
- C# introduces new access specifiers such as internal and protected internal

C# Vs C++

- C# has a concept called delegates which is similar to function pointers in C++
- Inheritance: In C++, classes and structs are identical
- No header files are used in C#. But namespaces are used extensively
- C# supports property, as opposed to traditional getter and setter methods in C++
- In C# everything including the Main() is written within a class

C# Vs C++

- Methods: C# supports four types of method parameters
  - Value parameters
  - Reference parameters
  - Out parameters
  - Parameter Arrays
  - Whereas in c++ its just pass by value and pass by reference

GAME DESIGN PATTERNS

Good Game Architecture

- Making Changes
- Decoupling
- Abstraction
- Simplicity
- Performance and Speed

Why Design Patterns?

Good design is like a refrigerator—when it works, no one notices, but when it doesn’t, it sure stinks

–Irene Au

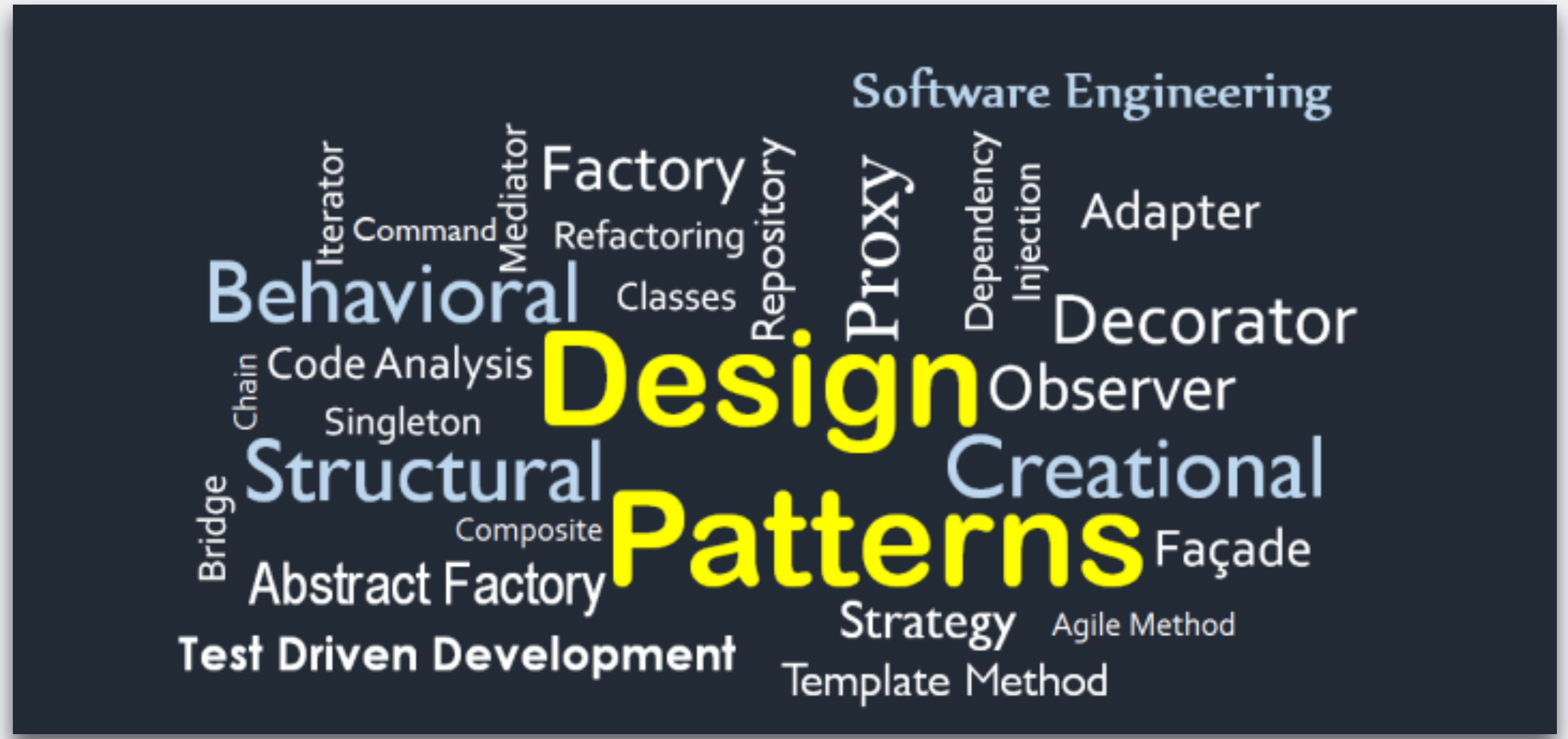
Why Design Patterns?

- No such thing as write once code
- Easier to understand
- Cleaner code
- Programming hygienes
- Develop faster?

What Are Design Patterns?

- Reusable solution to the commonly occurring problems
- Serve as templates to programmers
- Language independent
- Deals with Creation, Structuring / Behaviours

What Are Design Patterns?



BSC [**B**ehavioural **S**tructural **C**reational]

Classification

Creational

Singleton
Object Pool
Prototype
Factory

Behavioural

State
Command
Gameloop
Component
Observer

Structural

Adapter
Fly-weight

THANK YOU