Table Of Contents

ICT373Ans1 [MEMORY]

Software architecture:

- Fundamental organization of a system, embodied in its components, relationships between each other and the environment, and the principles
- Highest level view of the system that specifies the framework, languages, goals and methodology ~~which provides the blueprint for design, development and maintenance of software system~~
- Balances stakeholders needs such as marketing might a software that is short time to market and competitive advantage, customer wants low-cost product, organisation wants profit and leveraging existing assets
- Determines a set of constraints and requirements that must be followed during software design

- Documented and serves as a communication tool for development team, stakeholders, and users of the system

Software architecture patterns/styles are reusable solutions to common software design problems that are encountered in the development of complex software systems. These patterns are designed to provide a proven framework for designing, organizing, and implementing software systems that are scalable, maintainable, and robust. EG- Client-servicer architectures, Data abstraction and O-O organisation, Model-View-Controller (MVC) Pattern, Microservices Architecture Pattern-

Software architecture patterns/styles-

Pipe and filters architecture-

- Organizes the processing of data into a series of sequential filters
- Filter performs a specific transformation on the input data and passes it to the next filter through a communication channel called a pipe
- Pipe are connector of filters that transfers output of one filter to inputs of another
- Pipe may be type restricted such as integers only or string only

Advantages:

- Reuse: just pipe the filters together for all sorts of different applications
- Maintenance and enhancement: easy to add new filters or replace old ones with better versions
- Supports concurrency since can do the job with parrel filters

Disadvantages

- No cooperation between filters thus accidental processing duplication
- Processing may be bounded in capacity due to filters (such as filter process 1 bit at time thus next one bit processing)
- Designer may be forced into batch processing design

Software design:

- Refers to the process of creating a detailed plan or blueprint for the development and implementation of individual components, modules and class of the software system
- Focuses on specific algorithms, data structures, and programming techniques used to implement the components
- Factors include- organisation of data (data), structure of the eventual code of the solution, and how activities fit into the structure (packaging)
- Considers/Aware the ongoing implementation of the project's methodologies and goal

- Important aspect of software development process but address different abstraction level for instance in software design the organisation of the data is important

Predictive model:

- Described as a plan-driven, sequential approach to software development
- Requirements are assumed to be well defined and documented at the start
- Lack of customer engagement during because they are involved at the start and end
- Cost & Schedule of software development can be relied on
- Models include- waterfall and prototype

Adaptive model:

- Adaptive model can be described as incremental and iterative approach to software development
- Requirements are not set in stone and is assumed to might change during development process
- Greater customer engagement because they are given the opportunity to see iterative improvements and provide feedback
- Difficult to rely on the initial cost and schedule for the development
- Models include- scrum and extreme programming

Extreme programming:

- Programmers code in pairs and must write the tests for their own code
- Adhere to a set of principles such as test driven-development and continuous integration
- Focuses on whole team approach and collective code ownership
- XP teams include programmers, customers (manages the priorities to be done), architects, and managers

Scrum software development:

- Software is development occurs in short, fixed iterations known as sprints that two weeks to one month long
- Focus on optimising management and delivery of the project and teamwork
- Reliant on meetings such as sprint planning meeting and sprint review to ensure project is progressing as planned
- Scrum teams include- Scrum Master, Product Owner (managed product backlog), and the Development Team

Waterfall lifecycle model:

- Where software development occurs sequentially through a sequential series of well-defined stages
- Best suited for projects with well-defined requirements since after the requirement gathering there is little to no involvement from end-users or other stakeholders throughout the development process
- Lack of continuous improvement since at end stage the final product is delivered in full, which means that there is little opportunity for continuous improvement or iterative development
- Phases contains- requirements gathering, functional specification, design, implementation, testing, and maintenance

Prototyping model:

- Involves creating a model (a working replica) of the system to be constructed but the model is missing some of the essential functionality
- Requires heavy user involvement to clarify user requirements for operational software.
- Good because developers can easily throw away or keep prototypes depending on the project
- Reduce the risk of creating a system that doesn't meet user requirements or is too expensive/unreliable

TODO:

- RAD/SPIRAL

Note: LEARNING OBJECTIVES DON'T MATCH EXAM. EXPLAIN WHAT SOFTWARE ARCHITECTURE IS AND DESCRIBE

Lecture is where you COPY + PASTE AND UNDERSTAND HALF

STUDY (memorize) is where you reword + understand

ICT373Ans2 [MEMORY]

Software architecture patterns/styles-

Client-server architecture:

- Common architecture where client application(s) communicates with a server application over a network
- Client application(s) sends requests to a server application and the server responds to those request
- Servers are central repository of information which contains- the info to be updated, the software to manage the information and manage the distribution of information
- A centrally located server makes information easier to be changed and ensures information is latest
- Each client has software that communicates with the server, fetches and processes the information, and displays it for the user of the remote machine on their remote machine often via client's browser

Issues:

- Server side may be overloaded/congested BUT clients want error-free and fair processing thus consider simultaneous transaction processing
- Need to support multiple client machines types (Phone/mobile/Desktop) and multiple different of client operating systems types (Mac/Linux/Android)
- Need to ensures supporting software changes does not affect the continual compatibility for client
- Performance issue if there are many clients involved thus server processing needs to be very efficient

HTTP:

- Enables client-server model that allows multiple pages of information (containing text, pictures, sound, video) with links to other pages, set up on a server, to be read by a client (anywhere else on the Internet) using the web browser
- Basic HTTP Client-server system operations-
    - Client sends request to server
    - Server responds with a file/information
    - Client browser interprets and display the file/content
- Ensuring this works-
    - Client machines need a standard language (protocol) for defining the format of information so it can be sent by server (HTML)

    - Browsers also need to be able to interpreting the language on various client machine/operating system

Method for client communications with server- (HTTP Method provide a mechanism for communication/type of communication)

HTTP POST

- Submits data to be processed to a specified resource
- More secure since (HTTP Post) form submission means form data appears within the HTTP request message body not URL
- No data size limit for request so The HTTP POST message can send all sorts of information (text, sound, video) via an encoding to ASCII

HTTP GET:

- Requests data from a specified resource but can be used to pass information to a program on a server
- Less secure since GET method form submission = form data is encoded into the URL and appended to the action URL as query string parameters
- Faster since GET requested are cached by the browser thus faster load times for subsequent requests

Form validation-

Batch Validation

- Form is checked only when the user presses the submit button eg- onSubmit
- Stop the actual form submission to server side
- Client-side batch processing is faster than server side and allow errors to appear without the need of loading a new page

Real-time/run-time validation:

- Form is checked as individual events occur eg- onChange/onClick
- Easy for a user to bypass the real-time error since it's warnings
- Feedback is provided throughout the user filling the form

Client-side programming:

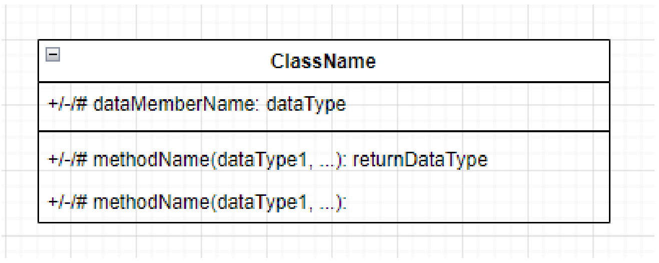
- Allow browser to run programs and allowing the server to send programs along with the HTML page
- A simple mistake on the form results in information needing to be is sent to the server, checked, and an error information page needing to be prepared and sent back
- Browser using JS can handle this and save server resource

TODO:

- URL, HTML, MIME
- Client-side programming VS server-side programming

ICT373Ans3 [1 day*(Priority)]

UML class: NOTE: Protected for derived



ClassName
+/-/# dataMemberName: dataType
+/-/# methodName(dataType1, ...): returnDataType
+/-/# methodName(dataType1, ...):

Cardinality:

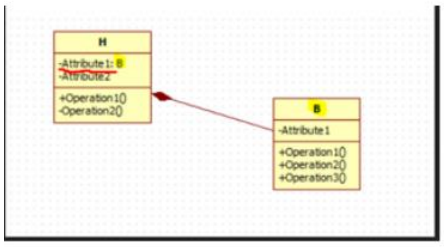
- 1 (Can have exactly 1)
- * (Can have zero or many)
- 0..1 (Zero or One)

UML relationships-

Association: relationship between classes can be drilled down to composition or aggregation not inheritance

Composition: (Parent class/wallet has a child class/money)

- [LOGICAL DIFFERENCE] When the object (parent class/wallet) disappears all the parts (child class/money) disappears.
- [IMPLEMENTATION DIFFERENCE] Why? Child object does not have their own life cycle.
- Class contains objects of another class. And the other class defines and implements certain behaviours. This class can thus reuse the other classes defined behaviour rather than writing from scratch
- Depends on parent's object lifecycle (to instantiate, and die)
    - Think created/instantiated or destroyed in parent class/constructor
- The parent class* has member variables (instance variables) that is the datatype of child
    - For example- When money is destroyed coins is destroyed



NOTE: Think "There I am" with hand/finger pointing

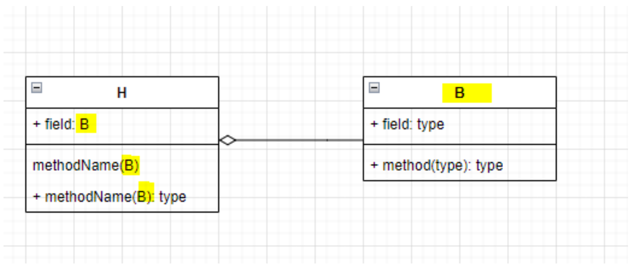
NOTE: Arrow is @ parent

NOTE: Can be List <B> listOfB as long as it is created in parent constructor. The entire list can't be replaced

Aggregation: (has a)

- [LOGICAL DIFFERENCE] When the object (parent class/Class) disappears all the parts (child class/student) DOESN'T disappear
- [IMPLEMENTATION DIFFERENCE] Why? Child object may live on even if parent is destroyed
    - Think data member is not constructed inside parent class. So a List<B> listB constructed is still composition. But if You have setters for data member or Constructor that takes an already constructed list then it is. I.E ASK who creates the data member object

- The parent class* has member variables (instance variables) that is the datatype of child



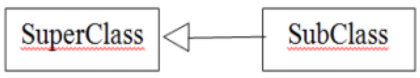
NOTE: Think "There I am" with hand/finger pointing

NOTE: Arrow is @ parent

NOTE: Constructed outside H

Inheritance:

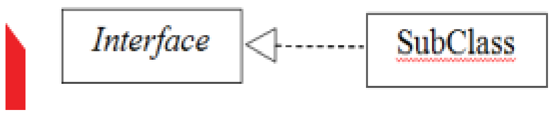
- Is a relationship where a class has all the characteristics of another class but then adds information
- Must not break L in SOLID
    - For: can relate them in English using x 'is a' y in a behavioural sense
    - For: The child class (cat) <u>uses</u> *every* member variable declared in the parent class (animal)
    - For: The child class (cat) <u>requires</u> *every* method defined in the parent class (animal)* Links to above in that if data member is not relevant setter is not required
- New classes can be created based on existing classes inheriting their public properties and methods thus w/o the need to rewrite the code from scratch



(Interface) Implementation:

- Defines a contract (functionality) that classes must implement
- What object can do
- Interface class contains no method implementation
- Interface can't contain any instance data members

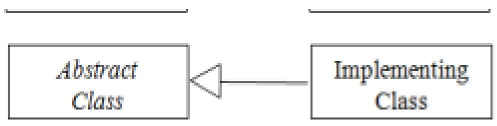
- Class can implement interface



Abstract class (Inheritance):

- Refers to an Is a relationship inheritance but a class has no logical reason to be instantiated. For example- Animal class is an abstract notion that is best served to be abstract class
- Abstract Class contains some method implementation
- Abstract Class can contain fields/data members

- Class can inherit abstract class



Java objects:

- Instance of a Java class that has identity, behaviour, state
- Out of scope variable may mean loss of reference to object

Garbage collector:

- Called when object no longer plays any part in the program
- Releases the memory for that object w/o programmers explicitly reclaiming
- May require certain actions to occur before releasing such as closing file

Call by value / reference:

- Primitive types are passed by value to the method
- Class type(s) are passed by value to the method but value is reference to an object or address of the object
    - Changes to the object attributes

    - Can’t change the reference of the object passed i.e Reassigning Object parameters doesn't affect the argument

Static members:

- Members belonging to the whole class and all objects have access to that same data member/method eg- Accessed className.field and ClassName.Method

Wrappers:

- Represent values of primitive types as class objects
- Useful as some container classes are only able to contain objects
- Boxing is conversion of value of primitive to corresponding class object while unboxing is conversion from object of wrapper class to its corresponding primitive type

Library Classes

- Ready-built classes that supply well designed data structures or help with input and output, networking, and GUIs i.e

Object Oriented Software Engineering:

ICT373Ans4 [MEMORY]

Abstraction:

- Ensures the client is dealing with the essential features of something while ignoring the implementation detail
- Seeks to provide a human understanding of how data structure should behave

Encapsulation:

- Restricting of direct access to some of an object's component

Information-hiding:

- Hiding the internal implementation details from client implementation level
  - For example- An average calculation method has an private sum of all values

Java features for reuse-

Methods:

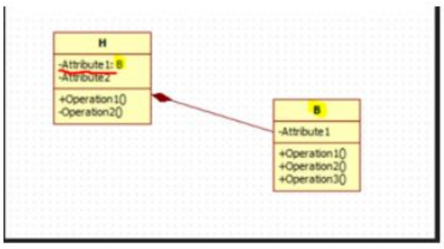
- Encapsulates a specific set of operations/behaviours that can be used repeatedly throughout a program without the need to rewrite the code each time

Packages:

- Collection of related classes grouped together in a directory
- Related public classes in the package can be easily imported for use by other classes

Composition: (Parent class/wallet has a child class/money)

- [LOGICAL DIFFERENCE] When the object (parent class/wallet) disappears all the parts (child class/money) disappears.
- [IMPLEMENTATION DIFFERENCE] Why? Child object does not have their own life cycle.
- Class contains objects of another class. And the other class defines and implements certain behaviours. This class can thus reuse the other classes defined behaviour rather than writing from scratch
- Depends on parent's object lifecycle (to instantiate, and die)
  - Think created/instantiated or destroyed in parent class/constructor
- The parent class* has member variables (instance variables) that is the datatype of child
  - For example- When money is destroyed coins is destroyed



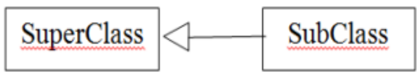
NOTE: Think "There I am" with hand/finger pointing

NOTE: Arrow is @ parent

NOTE: Can be List <B> listOfB as long as it is created in parent constructor. The entire list can't be replaced

Inheritance:

- Is a relationship where a class has all the characteristics of another class but then adds information
- Must not break L in SOLID
  - For: can relate them in English using x 'is a' y in a behavioural sense
  - For: The child class (cat) uses *every* member variable declared in the parent class (animal)
  - For: The child class (cat) requires *every* method defined in the parent class (animal)* Links to above in that if data member is not relevant setter is not required
- New classes can be created based on existing classes inheriting their public fields and methods thus w/o the need to rewrite the code



Collection: (Java collection framework)

Overloading:

- Occurs when there are multiple methods in a class with the same name but different parameters
- Overloaded methods must have a different number or type of parameters
- Uses static binding/static polymorphism/compile time

Overriding:

- Occurs when a subclass provides a specific implementation of a method that is already defined in its superclass
- Overrided method in the subclass must have the same parameters, name and return type as the method in the superclass
- Uses dynamic binding/dynamic polymorphism/runtime

Dynamic binding:

- Runtime binding mechanism that determines the method/variable to be called at runtime based on the type of object
- Enables polymorphism, which is the ability of objects of different types to be treated as if they are same type. Thus, objects can be tricked into calling a method on their base class, even if the method is overridden
- Binding of virtual methods is determined at runtime

Static binding:

- Compile-time binding mechanism that determines the method/variable to be called at compile-time based on the type of the reference variable
- Enables Early Error Detection because allows errors to be detected at compile-time rather than at runtime
- Binding of static/private/final methods is determined at compile-time

Inner class:

- A class declared within another class (nested classes)
- Allows for the containing class to implement behaviour of outer class while encapsulating the details in the containing class hence promoting information hiding and name management
- Inner classes have special access to the variables/fields of their containing classes
- Bypasses the need to create a new file for every new type

~~Callback function:~~

- ~~Object is given a piece of information that allows it to call back into the originating object at some later point~~
- ~~Useful for async i.e pass a function or method as an argument to another function, which can then be called back at a later point in time~~

~~Final:~~

- ~~Final class means it can’t be inherited by another class~~
- ~~Final method means can’t be overridden~~

TODO:

- ~~Inner class (1 Day)~~
- ~~Call back functions (1 Day)~~

- Collections + Generics
- ~~Downcasting VS upcasting~~
- ~~Private VS Protected method/Access modifiers~~
- ~~Final + Exception~~
- Extended abstract vs interface

Java features for encapsulation: //Maybe

- Packages
- Access modifiers

Composition:

Inheritance:

Packages:

Collection: (Java collection framework)

separate objects

Abstract class:

Compostion:

https://softwareengineering.stackexchange.com/questions/235313/uml-class-diagram-notations-differences-between-association-aggregation-and-co

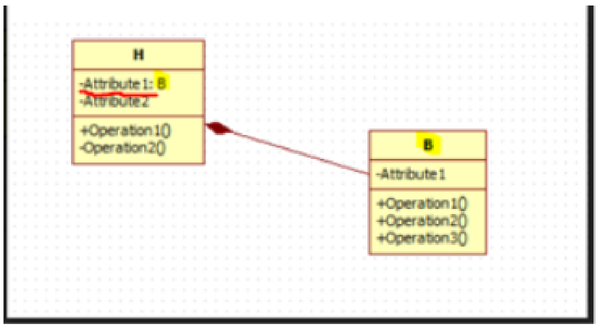
https://www.c-sharpcorner.com/UploadFile/ff2f08/association-aggregation-and-composition/

https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition

https://softwareengineering.stackexchange.com/questions/61376/aggregation-vs-composition

https://stackoverflow.com/questions/11881552/implementation-difference-between-aggregation-and-composition-in-java

Direction of array. Think “There I am” with hand/finger pointing



Arrow points to parent (bigger class)

ICT373Ans5 [MEMORY]

Persistence of objects:

- Ability of objects to retain their state beyond the lifetime of the program that created them. For example- Games storing/saving the state of the game so that it can be resumed later. For example- Financial applications storing data about transaction history and account information
- Implemented via serialisation

Serialization:

- Process of converting an object into a stream of bytes that can be saved to a file or sent over a network
- Allows an object (at a particular moment) to be kept or sent somewhere (eg, across a network) and then later restored to its previous state

Process of serialization: (Json.stringify)-

Deserialization: (Read) [ID]

```
FileInputStream txtInputFile = new FileInputStream("student.txt");

ObjectInputStream in = new ObjectInputStream(txtInputFile);


ClassName tempStudent = (ClassName) in.readObject();


txtFile.close();

in.close();
```

Serialization: (Write to file) NOTE: Class must implements Serializable

```
FileOutputStream txtOutputFile = = new FileOutputStream("student.txt");

ObjectOutputStream out = new ObjectOutputStream(txtOutputFile);


out.writeObject(ObjectName);


txtOutputFile.close():

out.close():
```

Importance of Serialization:

- Allows objects to kept in a standard format and transferred easily over a network in a standard format. For example- JSON and the internet
- Enables Remote Method Invocation (RMI) which allows Java objects to be invoked across different machines on a network (communication between objects via sockets)
- Enables Cross JVM Synchronization which is Java Virtual Machines (JVM) running on different architecture

Run-time type identification (RTTI):

- Ability of a program to work out the type of object at run-time. For example- it can be for finding the specific child class when you have parent object
- Why? Incorrect down casting produces runtime error. Use polymorphism

Run-time type identification via Instanceof:

```
for(Animal tempAnimal : animalList) {

       if(tempAnimal instanceOf Cat) { //Check whether object is instance of Cat

            Cat tempCat = (Cat) tempAnimal; //Downcast to correct object avoiding runtime error

            catList.add(tempCat);

       }

}
```

Reflection:

- Need more information about the type thus requires additional package java.lang.reflect
- Works out class information (e.g. fields and methods) of objects at run time
- RTTI when all relevant classes are not available at compile

```
for (Customer tempCustomer: customerList) {

            Class<?> tempObject = tempCustomer.getClass();

            if (tempObject.getSuperclass() == Cashier.class) {

                System.out.println("Child object: " + tempCustomer);

            }

}
```

```
// If the superclass is equal to Child.class, we know that the object is an instance of the
Child class
```

Object cloning is the process of creating a new object with the same state as an existing object. There are two types of object cloning: shallow copy and deep copy

Shallow copy:

- Creates a new object that shares all or some of the original object's instance variables and their referenced mutable object
- Any changes to the referenced mutable object(s) is reflected in the other reference as well (think class has OBJECT data type fields)

- Implemented using default clone by implementing cloneable interface + override clone method with super.clone()

```
public Object clone() throws CloneNotSupportedException {

        return super.clone();

 }

//Client

Employee original = new Employee(1, "Admin", hr);

Employee cloned = (Employee) original.clone();
```

Deep copy:

- Creates a new object that is completely separate from the original object and its referenced mutable objects
- Changes to one mutable objects reference is NOT reflected in the other reference

- ~~Deep copy can be implemented using serialization or by implementing a custom clone() method that performs a deep copy~~

Deep copy (cloneable): [Think overridden equals]

- Implement cloneable interface + override clone method (THINK override equals) with super.clone() with downcasted
- ~~go through each member field (especially mutable) in turn, get a (deep) clone of the component object and use that as the corresponding component of the new object~~

NOTE: Applies to classes with mutable object (parent)

```
public Object clone() throws CloneNotSupportedException {

       ThisClassName copy = (ThisClassName) super.clone();

                      //Go through each member field (only mutable object since above mem)

          copy.Field1Name = (Field1ClassName) this.Field1Name.clone();


        return copy;

  }

//CLIENT

Employee original = new Employee(1, "Admin", hr);

Employee cloned = (Employee) original.clone();
```

- Implement this class on all the subclasses

```
public Object clone() throws CloneNotSupportedException {

        return super.clone();

 }
```

Deep copy (Copy constructor):

- Copy constructor approach is where is a (copy) constructor takes an object of the same class as a parameter and initializes a new object with the same values as the parameter object
- ~~For inheritance there may be issues~~

```
    // copy constructor

    public Person(Person newPerson) {

               //Go through each member field

        this.name = newPerson.name;

        this.address = new Address(newPerson.address);

    }
```

Inheritance-

```
public Man(Man newMan) {

               super(newMan);

               //Go through each member field

        this.fieldName = newMan.fieldName;

    }
```

Deep copy (Serialization):

- Serialize an object into a stream (eg, into a buffer or file) and deserialize the stream back into an object. This will result in a deep copy of the object.
- Slow and expensive way of cloning

Immutable objects:

- Objects whose state (i.e., their data) cannot be changed after creation
- Requires that fields are final and that no public methods change the object

- Security: Immutable objects are inherently more secure than mutable objects, as they cannot be modified after they are created. This can help prevent malicious code from modifying the object's state and introducing security vulnerabilities

I/O in Java:

- OPENING: creating a stream object for each input source or output destination
- LOOPING: getting values in or sending values out by calling methods on the stream object
- CLOSING: the file or connection by calling stream object close method

Decorator Pattern Approach:

- All the combinations of choices needed in managing a particular IO operation is vast
- Allows a basic stream object (input or output) to be successively decorated by layers of extra facilities

Text Files:

- Stores characters, one at a time, (2 bytes each)
- Fast/efficient (buffer-at-a-time read and write)

Binary Files:

- Different types of values coded differently to maximize efficient use of space (eg, each integer takes 4 bytes)
- Slow/less efficient (byte-at-a-time read/write)

~~Byte-based (streams) super classes:~~

- ~~Read and write binary data~~
- ~~Programs use byte streams to perform input and output of 8-bit bytes~~

~~Character-based (streams) classes:~~

- ~~Read and write textual info~~
- ~~Contains- reader + writer abstract super class~~

https://howtodoinjava.com/java/cloning/a-guide-to-object-cloning-in-java/

ICT373Ans6 [MEMORY]

JavaFX Application Components-

Stage:

- A stage (a window) contains all the objects of a JavaFX application
- Invokes show() method to display contents of a stage usually a scene
- Holds the scene(s)
- Passed as stage object to start() method of application class

Scene:

- Represents the physical contents of a JavaFX application
- Contains all the contents of a scene graph

Scene graph:

- Tree like hierarchical structure of nodes is added to a Scene
- ~~Defines~~ order in which Nodes are drawn on the screen, positioned and arranged relative to each other
- The first Scene Graph is known as the Root node

Node:

- Single graphical element that can be added to a Scene
- Every Node in the Scene graph has a parent except root node

JavaFX application process:

- Prepare a scene graph with the required nodes
- Create a root node for scene graph
- Add nodes to scene graph
- Prepare a scene by ensuring a root node is associated to the scene
- Prepare the stage, attach scene and display the contents of the scene using the show() method as shown

JavaFX GUI Layout process:

Create node

```
TextField textField = new TextField();
```

The class of the required layout is instantiated

```
vBox vbox = new vBox();
```

Set properties of layout

```
vbox.setSpacing(10);
```

Add all created nodes to layout

```
vbox.getChildren().addAll(textField);
```

JavaFX Event:

- Whenever a user interacts with the application (nodes), an event is said to have been occurred
- Foreground Events generated as consequences of a person interacting with the graphical components in a GUI
- Background Events require the interaction of end user are known as background events

JavaFX Event Handling:

- Mechanism that controls the event and decides what should happen, when an event occurs

Phases of handling/processing events- (not all phases occur)

Event dispatch chain/route construction:

- Path from the stage to the source Node whenever event is generated
- Describes the order in which events are dispatched to nodes in the scene graph

Event capturing phase:

- Firstly, Root node of the application dispatches the event
- Secondly, Event travels to all nodes in the dispatch chain from top to bottom
- Allows any event filters registered on parent nodes to intercept and process the event before it reaches the target node for processing

Event bubbling phase: (Think event target phase)

- Event travels from the target node to the stage node (bottom to top)
- Allows event handlers registered on parent nodes of the target node to handle the event and potentially modify its behaviour before it reaches the root node

Methods for handling/processing events-

Event filters:

- Handles an event during the event capturing phase of event processing
- Filters enable the parent node to provide common processing for its child nodes or to intercept an event and prevent child nodes from acting on the event

- Consuming the event prevents any child node on the event dispatch chain from acting on the event

```
EventHandler<MouseEvent> tmpEventHandler = new EventHandler<MouseEvent>() {

        @Override

        public void handle(MouseEvent e) {

                //Code

        |

};

//Add filter to NODE

Circle.addEventFilter/removeEventFilter(MouseEvent.Mouse_Clicked, tmpEventHandler);
```

Event handlers: (Handles events that are generated by the target node)

- Invoked during the target phase/bubbling phase of event handling
- EventFilter is executed (act on the event) before the EventHandler because capturing phase generally occurs first
- Consuming the event in an event handler stops any further processing of the event by parent handlers on the event dispatch chain

```
EventHandler<MouseEvent> tmpEventHandler = new EventHandler<MouseEvent() {

        @Override

        public void handle(MouseEvent e) {

                //Code

        |

};

//Add hander to NODE

Circle.addEventHandler/removeEventHandler(MouseEvent.mouse_Clicked., tmpEventHandler);
```

Call-back function:

- Refers to function that is passed as a parameter to another function and is called by that function at a later time
- For event handling such that call back functions to handle events like mouse clicks

GUI behaviour involves-

State:

- Represents a situation, during the operation of a system, in which it meets some condition, performs an action, or waits for some event occurrence

Transition:

- Represents the relationship between two states denotating/indicating that the system/object must perform an action in order to transfer from one state to another

Finite State Machine: (Ideal GUI design)

- GUI Design that produces an system or component design that can be checked for correctness of its behaviour
- Fixed finite set of states and transitions
- Can only be in one state at a time
- Each transition has a source state & destination state & input and maybe output

FSM criteria:

- Consistency/determinism for given state & input there is only one state that machine can move to
- Completeness means for given state and input there is an appropriate transition
- Reachability means every given state has a path to it from the start state and a path from it to an end state

TODO:

- Add Java code for JavaFX event handling/event filters

- Create layout (maybe)
- **Meally machine:**
- Output depends on the current state and the inputs applied to the machine
- Outputs are associated with transitions and are produced only when a transition occurs
- **Moore machine:**
- Output depends only on the current state of the machine
- Outputs are associated with states and are produced as soon as the machine enters a new state

ICT373Ans7 [MEMORY]

Process:

- Self-contained running program
- Hold multiple threads

Thread:

- Separate independently running subtask within a process
- Has own flow of control meaning own resources such as stack and variables
- Allows program work while at the same time waiting for and/or monitoring one or more events/inputs from the outside world (user, network connections or peripheral devices)
- ~~Enable concurrent execution of multiple units of work within a single Java program~~

(Multi) Threading (Java Methods to create new threads) -

Subclass of thread (Help create + manage thread hence enable multithreading)

- Define a class that extends the Thread class + overrides the run() method
- Reduces flexibility because Java doesn't support multiple inheritance hence inability to extend any other additional class which you require
- Each new thread has its own unique object associated with it, which is separate from the objects associated with other threads

```
public class SumOfArrayCalculator extends Thread {
    @Override
    public void run() {
        // code to be executed in the new thread
    }
}
//Client
SumOfArrayCalculator obj = new SumOfArrayCalculator(tempArray);
obj.start();run() method contains the code that will be executed in the new thread when the start() method is called
```

Runnable interface (Help create + manage thread hence enable multithreading)

- Define a class that IMPLEMENTS the Runnable interface and provides an IMPLEMENTATION for the run() method
- Increased flexibility and allows for better code reuse since other interfaces can be implemented and can extend other class
- Multiple threads can access and execute the same object run() method thus share same data or resources (Downside need to synchronise)
- ~~Create an instance of the class and pass it to a Thread object constructor which will enable Thread object starts running the run() method of the Runnable object when the start() method is called~~

```
public class DownloadFile implements Runnable {
    @Override
    public void run() {
        // code to be executed in the new thread
    }
}

//Client
DownloadFile file = new DownloadFile();
Thread thread = new Thread(file);
thread.start();
```

Synchronization: (wait/await):

- For multithreaded program- it is where thread(s) wait for some others thread(s) to do something before continuing (async)
- Avoid collisions which is conflicts/inconsistencies in program's behaviour due to threads trying to access/modify shared resources concurrently
- Allows for sharing data in a multithreaded program by allowing one thread at a time exclusive access to code that manipulates the shared object
- ~~Prerequisite conditions may need to be reached before a certain event can occur~~

Synchronization Design/Process:

- Based on monitors and monitor locks where the monitor make sure that its object's monitor lock can be held by one thread at a time. The process is-
- Methods declared as synchronized require a thread with a lock of specified object prior to being allowed to execute the code
- Thread that enters such method grabs the lock (if it is available) and holds it until the code is fully executed
- Other threads are blocked and must wait for lock of the method to be available

Advantages of multithreading:

- Improved performance and concurrency by allowing concurrent multiple threads executing, enabling tasks executing in parallel
- Improved resource utilisation cos multiple threads (i.e use max resouces), ~~you can maximize the utilization of available processing power and efficiently utilize system resources~~
- Allows scalability design for Java applications ~~that can scale well with increasing workloads~~

Disadvantages:

- Difficulty of writing code (complexity)
- Difficulty of managing concurrency since must be aware of deadlock and racing
- ~~Slow down due to CPU overhead of thread management~~

Deadlock: (know diff between collisions)

- When two or more threads are blocked indefinitely due to waiting for a resource held by each other
- Cause programs to be unresponsive
- Java synchronization mechanisms (locks and monitor locks) manage concurrent access to shared resources thus reducing the risk of deadlocks
  - EG → Thread 1 gets resource 1 + thread 2 gets resource 2. Both (hold the resources) wait for other resource to be available hence deadlock

Multithreading in JavaFX: (thread sleep enables concurrency allowing other threads to execute)

- Scene Graph not thread-safe thus scene graph shouldn't be accessed/modified by multiple threads concurrently
- JavaFX adheres single-threaded model where JavaFX Application Thread must handle all Scene Graph changes ~~e.g., rendering the scene + updating the UI~~
- Other threads trying to access/modify Scene Graph may cause issues such as crashes or unresponsive UI

runLater Method:

- 'Platform.runLater(Runnable)' method allows for updates to the JavaFX UI from the background thread safely
- Offload time-consuming tasks to background threads while updating the UI in a thread-safe manner
- Runs the specified Runnable on the JavaFX Application Thread at some unspecified time in the future.

Task and worker:

- Task is an abstract class (in JavaFX) representing a unit of work that can be executed asynchronously on a background thread
- Task implement worker interface provides additional methods to monitor and handle (progress) background task
- Perform asynchronous tasks on background threads while communicating and keeping the UI responsive and updating it with the results as needed

Thread states (life-cycle)

New (born) state:

- Thread object created but not started

Ready ~~(Runnable)~~ state:

- Invoked by thread method start()
- Thread can be executed by the thread scheduler
- Also, other threads in program run concurrently

Running state:

- Thread given a processor and is running its task
- Process operating system (os) uses to work out which thread to run is called thread scheduling

Waited state:

- Running thread waits for another thread to perform a task
- Transition back to the running/ready state when another thread says to continue executing

Time waiting state:

- Running thread waits for a specified period of time now a sleeping thread

Terminated state:

- Running state enters the terminated state when it successfully completes its task
- System disposes the state

~~**Block state: (reword)**~~

- ~~Where running state attempts to perform a task that cannot be completed immediately and must until that task completes~~

TODO:

- Daemons
- Petri Nets (extended FSM)

ICT373Ans8 [3 DAYS]

Network programming:

Layered architecture:

ICT373Ans9 [MEMORY]

Design patterns:

- Proven architectures (high-level design decisions) for developing object-oriented software
- Provide a solution to a recurring specific design problem
- Identify and specify abstractions above the level of single components/classes/instances
- Provide a common vocabulary for communication among designers

- Divided in creational, structural and behaviour patterns
- Promote design re-use and help reduce the complexity of design process

Creational pattern-

Singleton design pattern:

- Ensure that a class has one and only one instance object, and provide a global point of access to it
- Implemented by having class with static object and make constructor private so no further instances can be made
- Preferred to global variable because they don't pollute the global namespace with unnecessary variables

```
Public class EventDatabase {

    private static EventDatabase instance;

    private EventDatabase() {


    }

    public static EventDatabase getInstance() {

        if (instance == null) {

            instance = new EventDatabase();

        }

        return instance;

    }
}
```

Structural pattern-

Composite design pattern:

- Treat individual objects and groups of objects uniformly by composing objects into tree structures [to rep part-whole hierarchies]
- Allow clients to ignore difference between individual objects and compositions of objects
- Works consistently with hierarchical structure

Adapter design pattern:

- Converts the interface of a class into an interface that clients expect
- Allows classes with incompatible interfaces to work together by providing a bridge between them
- Adapter class is the middle man that translates the requests from the client (via target interface) into a format that the target class can understand and process
- Promotes reusability by allowing existing classes to be used in new contexts or system

- Adaptee refers to the incompatible class client cannot interact with

Behaviour patterns-

Observer pattern:

- Establishes a one-to-many relationship between objects
- When the state of one object (the subject) changes, all its dependent objects (observers) will be notified and updated automatically
- Subject maintains a list of observers and notifies observers of any changes in its state
- Observers interested in the state of the subject and subscribed to get updates

Software Architecture design pattens-

Repository/Blackboard architecture:

- Shared blackboard is a central repository for information that facilitates collaboration between multiple knowledge sources
- Requires synchronisation to ensure consistency and avoid conflicts by controlling access to blackboard and regulate concurrent write operations

- Flexible in incorporating new knowledge sources or modifying existing ones
- Knowledge sources collaborate by reading from and writing to the blackboard

Interpreter Pattern:

- Provides a way to evaluate and execute structured expressions or languages
- Used to define a domain-specific language and interpret expressions of that language
- Decouples the grammar and interpretation logic thereby simplifying implementation of a language
- May lead to excessive classes if frequent changes to the language structure