**REACT**

1. **What are React Life cycle methods?**

**componentWillMount** is executed before rendering, on both the server and the client side.

**componentDidMount** is executed after the first render only on the client side. This is where AJAX requests and DOM or state updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with delayed execution such as setTimeout or setInterval. We are using it to update the state so we can trigger the other lifecycle methods.

**componentWillReceiveProps** is invoked as soon as the props are updated before another render is called. We triggered it from setNewNumber when we updated the state.

**shouldComponentUpdate** should return true or false value. This will determine if the component will be updated or not. This is set to true by default. If you are sure that the component doesn't need to render after state or props are updated, you can return false value.

**componentWillUpdate** is called just before rendering.

**componentDidUpdate** is called just after rendering.

**componentWillUnmount** is called after the component is unmounted from the dom. We are unmounting our component in main.js.

1. **What is a virtual DOM and what is the significance of this?**

In React, for every [DOM object](http://eloquentjavascript.net/13_dom.html), there is a corresponding "virtual DOM object." A virtual DOM object is a *representation* of a DOM object, like a lightweight copy.

A virtual DOM object has the same properties as a real DOM object, but it lacks the real thing's power to directly change what's on the screen.

Manipulating the DOM is slow. Manipulating the virtual DOM is much faster, because nothing gets drawn onscreen. Think of manipulating the virtual DOM as editing a blueprint, as opposed to moving rooms in an actual house.

1. **In what all life cycle methods can we set the state and why not in others?**

We can set the state in all life cycle methods except componentWillUpdate. In componentWillUpdate we can’t set the state because if you set the state in componentwillupdate, render will go in to infinite loop. **componentWillUpdate** is called just before rendering.

1. **Which was the new life cycle method added as part of react 16?**

To make class component an error boundary we define a new lifecycle method called *componentDidCatch(error, info).*

1. **Is force updation of react componenets good or not good ,explain**

Actually Force update is not recommended, force update will force fully call the render method without callilng the shouldcomponentupdate. It’s recommended to use this.setState and this.setProps methods instead of force update.

**REDUX**

1. **What are Smart Components and Dumb(Pure) Components?**

**Dumb Components**

Dumb components are also called ‘presentational’ components because their only responsibility is to present something to the DOM. Once that is done, the component is done with it. No keeping tabs on it, no checking in once in a while to see how things are going. Nope. Put the info on the page and move on.

The components themselves only have a render() method (they don’t need any others) and are often just Javascript functions. They don’t have internal state to manage. They wouldn’t know how to change the data they are presenting if they were asked. Ignorance is bliss.

**Smart Components**

Smart components (or container components) on the other hand have a different responsibility. Because they have the burden of being smart, they are the ones that keep track of state and care about how the app works.

Using the container design pattern, the container components are separated from the presentational components and each handles their own side of things. The container components do the heavy lifting and pass the data down to the presentational components as props.

They are class-based components and have their own state defined in their constructor() functions.

1. **What is the significance of mapstateToProps and  mapDispatchToProps?**

mapStateToProps : This function does exactly what the name suggests, connect state from the store to corresponding props. This make it possible to access your reducer state objects from within your React components. This function will subscribe to the Redux store and any updates will update props automatically. mapStateToProps needs to return an object, where the key is the new prop name to be used in the React app and the value is the name of the reducer function.

mapDispatchToProps: this can either be a function or an object. Now assuming that you have some knowledge of Redux, you’ll understand the importance of using ‘dispatch’. If not, here is a basic definition of ‘dispatch’ [from Redux](http://redux.js.org/docs/api/Store.html#dispatch):

Dispatches an action. This is the only way to trigger a state change.

The store’s reducing function will be called with the current getState() result and the given action synchronously. Its return value will be considered the next state.

1. **What are Reducers and Root Reducers?**

Reducer: any function with the signature (state, action) -> newState (ie, any function that couldbe used as an argument to Array.prototype.reduce)

Root reducer: the reducer function that is actually passed as the first argument to createStore. This is the only part of the reducer logic that must have the (state, action) -> newState signature.

1. **What is a middleware why do we need it?**

A middleware is simply a function with the signature

Middlewares add a nice encapsulation for store behaviour that does not form part of the data, and can also make testing a lot easier.

**JAVA**

1. **What is JVM and is it platform independent?**

Java Virtual Machine (JVM) is the heart of java programming language. JVM is responsible for converting byte code into machine readable code. JVM is not platform independent, thats why you have different JVM for different operating systems. We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It’s called virtual because it provides an interface that doesn’t depend on the underlying OS.

1. **What is the difference between JDK and JVM?**

Java Development Kit (JDK) is for development purpose and JVM is a part of it to execute the java programs.

JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence.

1. **What is the difference between JVM and JRE?**

Java Runtime Environment (JRE) is the implementation of JVM. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn’t contain any development tools like java compiler, debugger etc. If you want to execute any java program, you should have JRE installed.

1. **Which class is the superclass of all classes?**

java.lang.Object is the root class for all the java classes and we don’t need to extend it.

1. **Why Java doesn’t support multiple inheritance?**

Java doesn’t support multiple inheritance in classes because of “Diamond Problem”. To know more about diamond problem with example, read [Multiple Inheritance in Java](https://www.journaldev.com/1775/multiple-inheritance-in-java).

However multiple inheritance is supported in interfaces. An interface can extend multiple interfaces because they just declare the methods and implementation will be present in the implementing class. So there is no issue of diamond problem with interfaces.

1. **Why Java is not pure Object Oriented language?**

Java is not said to be pure object oriented because it support primitive types such as int, byte, short, long etc. I believe it brings simplicity to the language while writing our code. Obviously java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.

As we know, for all the primitive types we have wrapper classes such as Integer, Long etc that provides some additional methods.