Agile

* Agile: Deliver a basic product first, then build upon the basic product, deliver the minimum product first (MVP or Minimum Viable Product), Delivers products incrementally and frequently(weeks).
* Sprints: Last 2 weeks in which you add features to the product.
* Gives the client more input in the final product, change is welcome.
* Build only what is essential.

Key Terms:

* User Stories: Product Features, Requirements, or Tasks that add value to the end customers. Should be independent, negotiable, valuable, estimable, sized appropriately, testable.
* Story Points: Story points are a relative unit of measure to assess whether a user story is big or small.
* Product Backlog: Ordered list of everything that might be needed in the product.
* Sprint Backlog: The set of Backlog items selected to be delivered over a period of time.
* Velocity: The number of story points delivered over a Sprint.
* Swimlanes: Visual representation of stories
* MVP – Minimum Viable Product: Bare minimum product that meets the client’s expectations. Includes all must haves and excludes all nice to haves. (Doesn’t even have to do everything you want right away)
* Release: Comprised of several iterations or sprints
* Sprints: a period of time in which the team will work on a defined set of user stories.
* Epic: A user story that is so large that it requires multiple sprints to complete.

Agile Team:

* Product Owner: End customers or client, define priorities and what is valuable.
* Scrum Master: facilitates team meetings, removes team impediments, makes sure the team remains on track.
* Delivery Team: People in charge of execution or that are also part of the project’s implementation. Defines what user stories are part of a sprint.

Tools:

* Burn Down Chart: Chart that shows the burndown of work. Shows the work remaining, velocity of the team, and the work done.
* Kanban/Agile Board: A visual/pull system to track work and progress in a logical/basic flow. To do > In progress > in review > done

Rituals:

* Sprint Planning: Meeting where the team defines which user stories they will work on in an upcoming sprint. Scrum Master and Delivery Team
* Daily Stand Ups: 15 minute daily meeting where people stand up near the Kanaban Board and share what they did yesterday, what they will do today, and blocks. Scrum Master and Delivery Team
* Sprint Review (Demo): Meeting where the team presents what they have delivered over a sprint to the product owner. Product Owner, Delivery Team and Scrum Master
* Retrospectives: Meetings done after sprints to review, revise and adjust. Delivery team and Scrum Master.

Sprint 0(Planning):

1. Define who is part of the team.
2. Discuss with team the best time for agile rituals.
3. Get physical tools ready (Jira).
4. Business Case, budget, key requirements.
5. Define Constraints
6. Define what days you will release, test, and sprint.

Java

Good to know commands:

* + System.out.println(); = print() or cout <<
  + Import java.util.Scanner for user input in console
  + Scanner scan = new Scanner(System.in)
  + Inheritance: uses the “extends” keyword, do not need a virtual function to override parent functions.
  + Constructors are the same as they are in C#
  + Protected: allows member functions/variables to be accessed even within the same package.
  + “null” for null in variables/objects
  + Static runs the same as C#
    - To access instance variables in a static method, static variables required
  + Object class, all classes created are inherited from the superclass Object.
  + Try{}catch(Exception e){} for error handling
  + Throws will throw an exception on an error.
  + Swing and AWT
    - Abstract Classes: Contain one or more abstract methods, which has no implementation.
    - This means that all classes that inherit the Superclass will have to implement the method of the abstract class.
    - You cannot make an object of an abstract class.
    - Abstract methods cannot have a body or implementation.
    - @Override for polymorphism
    - Interface: for methods that are all abstract

Spring Framework

Spring

* Used for building Java Apps (Like .NET for C#)
* EJB is complex and has poor performance.
* Spring 1.0 (2004)
* Became extremely popular in the industry.

Spring 5

* Needs Java 8 or higher.

Spring Core Framework

* [www.spring.io](http://www.spring.io)
* Java Plain Old Java Objects
* Dependency injection to promote loose coupling.
* Aspect-Oriented-Programming (AOP).
* Spring Projects: Modules built in on top of the framework, use what you need
* Core Framework
  + Core Container
    - Manages how beans are created, factory for creating beans.
    - Beans, Core, SpEL, Context
  + Infrastructure
    - Aspect Oriented Programming: add functionality to objects declaratively.
  + Data Access Layer
    - JDBC Helper Class
    - Reduces overall code by 50%
    - Object to relational mapping
    - JMS
  + Web Layer
    - All Web related Classes
    - Spring MVC framework
  + Test Layer
    - Supports Test Driven Development

Inversion Of Control: The approach of outsourcing the construction and management of objects

* Outsource to an object factory.
* Object Factory based on a configuration file will give the correct configuration
* Spring Bean
  + A Java object created by the spring container.
* Spring Container
  + Create and manage obj. (Inversion of Control)
  + Inject objects dependencies (Dependency Injection)
  + XML Config File: Old/Legacy
  + Java Annotations
  + Java Source Code
* Development Process
  + Configure Beans
    - <bean id=”alias” class”Actual class”> </bean>
  + Create Spring Container
    - ApplicationContext
    - ClassPathXMLApplicationContext context = new ClassPathXmlContext(“context.xml”)
  + Retrieve Bean from Container
    - Context.getBean(“alias”,Coach.class)
* Dependency Injection
  + Constructor Injection
    - Define Dependency interface and class.
    - Create a constructor in class for injection.
    - Inject dependency in xml.
  + Setter Injection
    - Create setter method in your class for injection.
    - Configure the dependency injection in the config file.
* Singleton
  + Spring container makes only one instance of the bean.
  + Cached in memory.
  + Everyone will share it.
* Java Annotations
  + Markers added to Java classes that contain metadata.
  + Example @Override.
  + Dev Process
    - Enable Compontent Scanning in configuration file.
    - Add @Component(“Bean ID”).
    - Retrieving the bean is the exact same as using the XML file.
    - Not putting a BeanID will give it a default beanID.
* AutoWiring
  + Constructor Injection
  + Setter Injection
  + Field Injection
  + @Autowired
  + Autowired will always run
* Qualifier
  + Use @Qualifier(“BEANID”)
* @Scope(“Scope”) for changing scope\
* @PostConstruct
* @PreDestroy
* Java Configuration (No XML)
  + Create a Java Class and annotate as @Configuration.
  + Use @ComponentScan(“package”) to scan the package.
* Spring MVC
  + Web apps using Java.
  + Leverage a set of reusable UI components.
  + A set of web pages
  + A collection of spring beans.
  + Spring configuration.
  + Make Model View Controller
* Front Controller
  + DispacherServlet.
  + Part of Spring Framework.
* Controller
  + Created by developer
  + Contains business logic, handles the request
  + @Controller for controller classes
  + @RequestMapping(“/”)
* Model
  + Container for the data
* View Template
  + Many different view templates (JSP)
* Validation
  + @Valid runs the validation conditions.
  + @InitBinder will preprocess the web request(removes whitespace)
  + You can create custom Java Validation
    - Use @interface
* Advanced Mapping(ER Model Concepts)
  + One to One: X can have one Y.
  + One to Many: X can have many Y.
  + Many to Many: Multiple X can have Multiple Y
  + Primary Key and Foreign Key
  + Cascade: Cascading operation, apply same operation to related entites. (This mainly Applies to One-to-One relationships, Especially Referential Integrety Constraints)
  + Eager: retrieve everything.
  + Lazy: will retrieve on request.
  + @OneToOne is unidirectional
  + @OneToMany(mappedBy = “othertable) is bi-directional
  + @ManyToOne
  + Needs @JoinColumn(name = “column in other table”)
  + This all relates to the database concepts for relations.
* Entity Lifecycle
  + Detach – not associated with Hibernate Session
  + Merge – Reattach to session.
  + Persist – Refresh
  + Remove – Delete, Commit, Rollback
* Cascade Types
  + PERSIST – if entity is saved, so is cascaded data
  + REMOVE – if enity is removed, so is the cascaded data
  + REFRESH
  + DETACH
  + ALL
  + By default no operations are cascaded, can list each one and add a comma.
* Data Access Object
  + Data Access Object needs a Hibernate SessionFactory
  + SessionFactory needs the datasource (how to connect to the database).
  + These are all dependencies
* Aspect Oriented Programming
  + Cross Cutting Concerns.
  + Aspects can be reused at multiple locations.
  + Aspect: module of code for cross-cutting
  + Advice: what action is taken and when it should be applied
    - Before Advice: run before the method.
    - After Advice: run after the method finishes.
    - After Returning Advice: run after success.
    - After Throwing Advice: run after exception thrown.
    - Around: run before and after
      * Good for runtime tests
      * ProceedingJoinPoint
  + Join Point: When to apply code during program execution.
    - @Before(“POINTCUT”)
  + Pointcut: A predicate expression for where advice should be applied.
    - “execution(public void runCode())” will run before runCode() is called
      * “execution(public void com.luv2code.aopdemo.dao.AccountDAO.addAccount())”
    - Can use wildcards \* and .. (“execution(public void add\*())”) will run for all add functions
      * (“execution(public \* runCode())”) will run a runCode with any return type since it has a wildcard.
  + Weaving: Connecting aspects to target objects to create an advised object.
* REST
  + Representational State Transfer
  + Make REST API calls over HTTP.
  + Commonly uses XML and JSON.
  + You can call an external API to get a JSON blob of what you need.
* JSON
  + “name”: value (name-value pairs)
  + Values
    - Numbers
    - String using “”
    - Boolean
    - Nested JSON object
    - Array []
    - Null
  + Data Binding to convert JSON to Java POJO (Java object)
* Spring Boot 3
  + Contains the server so don’t need to install anything.
  + IntelliJ better
  + It uses Spring, MVC, REST, etc.
  + Just makes it easier to get started.
  + Creates a Maven project.
    - Maven is a Project Management Tool.
    - Maven will download the project jar files for you, so you don’t have to manually add the jar files to each project
    - Maven will read the config file, check the local repository, else goes to the internet and save it in the local repository.
    - POM.xml file
      * Project Object Model (Config file).
      * Project Metadata
      * Dependencies
      * Plug ins
    - Mvnw
      * Maven wrapper file, automatically downloads the latest version of maven
    - Spring Boot Starters
      * Curated list of Maven dependencies
      * Collection of dependencies
  + CRUD
    - C = persist
    - R = find
    - U = find -> merge
    - D = remove
    - executeUpdate() will execute the modification of the database

Python

* Command Prompt
  + cd = current directory
  + dir = ls
  + cls = clear