Day-Wise Notes - Java & Spring Boot

Oviya P <oviya.p@hcltech.com>
Mon 11/27/2023 5:12 PM
To:Oviya P <oviya.p@hcltech.com>

Classification: **Confidential**

JAVA & Spring Boot – Entire Notes

Subject: RE: Reg: Day 25 Notes

Classification: **Confidential**

```
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;

import com.pack.SpringbootMockito.entity.Movie;
import com.pack.SpringbootMockito.repository.MovieRepository;
import com.pack.SpringbootMockito.service.MovieService;


@ExtendWith(MockitoExtension.class)
@SpringBootTest  //loads entire application context
public class TestMovieService {

	@MockBean
	MovieRepository movieRepo;

	@Autowired
	MovieService movieService;

	@Test
	public void testCreateMovie() {
		Movie movie=Movie.builder().id(1001).name("Nun").language("English").type("Horror").rating(4).build();
		when(movieRepo.save(movie)).thenReturn(movie);
		assertThat(movieService.createMovie(movie)).isEqualTo(movie);
	}

	@Test
	public void testGetAllMovies() {
		Movie movie1=Movie.builder().id(1001).name("Nun").language("English").type("Horror").rating(4).build();
		Movie movie2=Movie.builder().id(1002).name("Room").language("English").type("Horror").rating(3).build();
		List<Movie> list=new ArrayList<>();
		list.add(movie1);
		list.add(movie2);
		when(movieRepo.findAll()).thenReturn(list);
		assertThat(movieService.getAllMovies()).isEqualTo(list);
	}

	@Test
	public void testGetMovieById() {
		Integer id=1001;
		Movie movie=Movie.builder().id(1001).name("Nun").language("English").type("Horror").rating(4).build();
		when(movieRepo.findById(id)).thenReturn(Optional.of(movie));
		assertThat(movieService.getMovieById(id)).isEqualTo(movie);
	}
}
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.ArrayList;
import java.util.List;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.pack.SpringbootMockito.entity.Movie;
import com.pack.SpringbootMockito.repository.MovieRepository;

@ExtendWith(MockitoExtension.class)
@SpringBootTest
public class TestMovieRepository {

	@Autowired
	MovieRepository movieRepo;

	@Test
	public void testCreateMovie() {
		Movie movie=Movie.builder().id(1001).name("Nun").language("English").type("Horror").rating(4).build();
		Movie savedMovie=movieRepo.save(movie);
		Movie movie1=movieRepo.findById(savedMovie.getId()).get();
		assertEquals(movie.getId(),movie1.getId());
	}

	public void testGetAllMovies() {
		List<Movie> list=movieRepo.findAll();
		List<Movie> list1=new ArrayList<>();
		for(Movie m: list) {
			list1.add(m);
		}
		assertEquals(list.size(),list1.size());
	}
}
```

Subject: RE: Reg: Day 24 Notes

Classification: **Confidential**

1. Create springboot project with spring web, spring data jpa, h2 db, lombok, dev tool

2. Configure db in application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto= update

spring.h2.console.enabled=true
# default path: h2-console
spring.h2.console.path=/h2-ui
spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER
server.port=2000
```

3. Create entity class

```
@Entity
@Data
```

```
@AllArgsConstructor
@NoArgsConstructor
public class Movie {
  @Id
  private Integer id;
  private String name;
  private String language;
  private String type;
  private Integer rating;
}
```

4. Create repository

```
public interface MovieRepository extends JpaRepository<Movie, Integer> {
  List<Movie> findByType(String type);
}
```

5. Create controller

```
@RestController
@RequestMapping("/api")
public class MovieController {

	@Autowired
	MovieService movieService;

	@PostMapping("/movie")
	public ResponseEntity<Movie> createMovie(@RequestBody Movie movie){
		try {
			Movie savedMovie=movieService.createMovie(movie);
			return new ResponseEntity<Movie>(savedMovie,HttpStatus.CREATED);
		}
		catch(Exception e) {
			return new ResponseEntity<>(null,HttpStatus.INTERNAL_SERVER_ERROR);
		}
	}

	@GetMapping("/movie")
	public ResponseEntity<List<Movie>> getAllMovies() {
		try {
			List<Movie> list=new ArrayList<>();
			movieService.getAllMovies().forEach(list::add);
			if(list.isEmpty()) {
				return new ResponseEntity<>(HttpStatus.NO_CONTENT);
			}
			return new ResponseEntity<>(list,HttpStatus.CREATED);
		}
		catch(Exception e) {
			return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
		}
	}

	@GetMapping("/movie/{movid}")
	public ResponseEntity<Movie> getMovieById(@PathVariable("movid") Integer id){
		Movie movie=movieService.getMovieById(id);
		if(movie!=null) {
			return new ResponseEntity<>(movie,HttpStatus.OK);
		} else {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
	}

	@GetMapping("/movie/byType")
	public ResponseEntity<List<Movie>> getMovieByType(@RequestParam("type")String type){
    List<Movie> list=movieService.getMovieByType(type);
    if(list.isEmpty()) {
	return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(list,HttpStatus.OK);
	}

}
```

----Create Service

```
@Service
public class MovieService {

	@Autowired
	MovieRepository movieRepo;

	public Movie createMovie(Movie movie) {
		return movieRepo.save(new Movie(movie.getId(),movie.getName(),movie.getLanguage(),movie.getType(),movie.getRating()));
	}

	public List<Movie> getAllMovies(){
		return movieRepo.findAll();
	}

	public Movie getMovieById(Integer id) {
		return movieRepo.findById(id).get();
	}

	public List<Movie> getMovieByType(String type){
		return movieRepo.findByType(type);
	}
}
```

6. To give request we have to use either PostMan or Swagger it is a documentation tool where it will document all the request and provides an UI

```
<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.9.2</version>
		</dependency>
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger-ui</artifactId>
			<version>2.9.2</version>
		</dependency>
```

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

	@Bean
	public Docket movieApi() {
		return new Docket(DocumentationType.SWAGGER_2)
					.select()
					.apis(RequestHandlerSelectors.basePackage("com.pack"))
					.paths(regex("/api/movie.*"))
					.build();
	}
}
```

7. To open swagger
http://localhost:2000/swagger-ui.html

8. To open h2 console
http://localhost:2000/h2-ui

spring-boot-starter-test - used to write testcases in spring boot - comes with junit5, mocktio, hamcrest

```
<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
```

@WebMvcTest - loads only the web layer which includes security, filter, interceptors etc for handling request/response. It is mainly used to write test for @Controller or @RestController

MockMvc class is a part of Spring MVC framework which helps in testing controller by explicitly starting server, which will be used along with Springboot WebMvcTest class to execute Junit testcases which test controller prg

@MockBean - to add mock object to spring boot appl

@Builder - used to build the object - Builder design pattern

```
import static org.hamcrest.CoreMatchers.is;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.BDDMockito.given;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import java.util.ArrayList;
import java.util.List;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import static org.junit.jupiter.api.Assertions.*;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.pack.SpringbootMockito.controller.MovieController;
import com.pack.SpringbootMockito.entity.Movie;
import com.pack.SpringbootMockito.service.MovieService;

@ExtendWith(MockitoExtension.class)
@WebMvcTest(value=MovieController.class)
public class TestMovieController {

	@Autowired
	MockMvc mockMvc;

	@MockBean
	MovieService movieService;

	@Autowired
	ObjectMapper objectMapper;

	@Test
	public void testCreateMovie() throws Exception {
		Movie movie=Movie.builder().id(1000).name("Robert")
					.language("Tamil").rating(4).type("action").build();
		given(movieService.createMovie(any(Movie.class)))
		    .willAnswer((inv)->movie);
		ResultActions response=mockMvc.perform(post("/api/movie")
					.contentType(MediaType.APPLICATION_JSON)
					.content(objectMapper.writeValueAsString(movie)));

		 response.andDo(print())
			.andExpect(status().isCreated())
			.andExpect(jsonPath("$.name",is(movie.getName())))
			.andExpect(jsonPath("$.language",is(movie.getLanguage())));
	}

	@Test
	public void testGetAllMovies() throws Exception {
		Movie movie1=Movie.builder().id(1000).name("Robert")
					.language("Tamil").rating(4).type("action").build();
		Movie movie2=Movie.builder().id(1001).name("Lost world")
```

```
                .language("English").rating(4).type("action").build();

        List<Movie> list=new ArrayList<>();
        list.add(movie1);
        list.add(movie2);

        given(movieService.getAllMovies()).willReturn(list);

        ResultActions response=mockMvc.perform(get("/api/movie"));
        MockHttpServletResponse res=response.andReturn().getResponse();
        Movie[] mov=new ObjectMapper().readValue(res.getContentAsString(), Movie[].class);
        assertEquals("Robert",mov[0].getName());
        assertEquals("English",mov[1].getLanguage());

    }

    @Test
    public void testGetMovieById() throws Exception {
        Integer id=1000;
        Movie movie1=Movie.builder().id(1000).name("Robert")
                .language("Tamil").rating(4).type("action").build();

        given(movieService.getMovieById(id)).willReturn(movie1);

        ResultActions response=mockMvc.perform(get("/api/movie/{movid}",id));
        MockHttpServletResponse res=response.andReturn().getResponse();
        Movie m1=new ObjectMapper().readValue(res.getContentAsString(),Movie.class);
        assertEquals("Tamil",m1.getLanguage());

    }
}
```

Subject: RE: Reg: Day 23 Notes

Classification: **Confidential**

1. Create java maven project with 2 dependency junit-jupiter, mockito dependency

```
<dependencies>
    <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.5.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>3.2.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

2. Create model class

```
public class Book {
        private String bookId;
        private String title;
        private int price;
        private LocalDate publishedDate;
    //getter,setters and constructor
}
```

3. Create repo interface

```
public interface BookRepository {
        Book findBookByBookId(String bookId);
        void save(Book book);
```

```
}
```

4. Create service class

```
public class BookService {

	private BookRepository bookRepository;

	public BookService(BookRepository bookRepository) {
		this.bookRepository = bookRepository;
	}

	public int calculateTotalCost(List<String> bookIds) {
		int total = 0;
		for(String bookId : bookIds){
			Book book = bookRepository.findBookByBookId(bookId);
			total = total + book.getPrice();
		}
		return total;
	}

	public void addBook(Book book) {
		bookRepository.save(book);
	}


}
```

@RunWith(MockitoJunitRunner.class) - Junit4 will support annotation based mockito

@ExtendWith(MockitoExtension.class) - Junit5 will support annotation based mockito

controller - service - repostiory - database

@InjectMocks - It tells which class is under test - prg that is communicate with external dependency or class annotated @Mock

@Mock - create a mock implementation for the class u need or whatever external implementation we need to mock

1. Stubbing of methods
   - One of the primary benefit of mockito is ability to return a provided response when a specific method is called on mock dependency, so we need to tell mockito what to return as response of that method
   - The process of writing how a given mock method should behave or return that is called stubbing

2 ways
1. using mockito static method when()+thenReturn() - "when" any specific method is called on mock object, "thenReturn" some preconfigured value
2. using mockito static method doReturn()+when() - "doReturn" some preconfigured value, "when" specific method is called on mock object
3. test void methods, then we have to use doNothing()+when()

- Mockito uses equals() while matching argument during stubbing, in case response is not stubbed for a method, then default value are returned when called
  primitive datatype - return 0
  boolean - return false
  any object and array - return null
  any collection - return empty collection

```
@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

	@InjectMocks
	BookService bookService;
	@Mock
	BookRepository bookRepo;

	@Test
	public void testCalculateTotalCost() {
		List<String> bookIds=new ArrayList<>();
		bookIds.add("1234");
		bookIds.add("1235");

		Book book1=new Book("1234","Mockito",500,LocalDate.now());
```

```
		Book book2=new Book("1235","Junit5",400,LocalDate.now());

		//when+thenReturn
		/*when(bookRepo.findBookByBookId("1234")).thenReturn(book1);
		when(bookRepo.findBookByBookId("1235")).thenReturn(book2);
		int cost=bookService.calculateTotalCost(bookIds);
		assertEquals(900,cost);*/

		//doReturn+when
		/*doReturn(book1).when(bookRepo).findBookByBookId("1234");
		doReturn(book2).when(bookRepo).findBookByBookId("1235");
		int cost=bookService.calculateTotalCost(bookIds);
		assertEquals(900,cost); */

		//In case we didnt give proper parenethesis we get UnfinishedStubbingException
		/*doReturn(book1).when(bookRepo.findBookByBookId("1234"));
		doReturn(book2).when(bookRepo.findBookByBookId("1235"));
		int cost=bookService.calculateTotalCost(bookIds);
		assertEquals(900,cost); */

		//when(bookRepo.findBookByBookId("1234")).thenReturn(book1);
		//when(bookRepo.findBookByBookId("1234")).thenReturn(book1);
		/*when(bookRepo.findBookByBookId("1234"))
			.thenReturn(book1)
			.thenReturn(book1);
		int cost=bookService.calculateTotalCost(bookIds);
		assertEquals(1000,cost);*/
	}

	@Test
	public void testAddBook() {
		Book b1=new Book("1235","Junit5",400,LocalDate.now());
		doNothing().when(bookRepo).save(b1);
		bookService.addBook(b1);
	}
}
```

2. Exception Handling in Mockito
   - Exception covers our -ve scenario

- If we test exception for non void methods then we use when()+thenThrow()
- If we test exception for void methods then we use doThrow()+when()

1. Create model class

```
public class Book {
	private String bookId;
	private String title;
	private int price;
	private LocalDate publishedDate;
   //getter,setters and constructor
}
```

2. Create repo

```
public interface BookRepository {
	List<Book> findAllBooks() throws SQLException;
	void save(Book book) throws SQLException;
}
```

3. Create service class

```
public class BookService {

	private BookRepository bookRepository;

	public BookService(BookRepository bookRepository) {
		this.bookRepository = bookRepository;
	}

	public int getTotalPriceOfBooks() {
```

```
		List<Book> books = null;
		try {
			books = bookRepository.findAllBooks();
		} catch (SQLException e) {
			// log exception
			throw new DatabaseReadException("Unable to read from database due to - " + e.getMessage());
		}
		int totalPrice = 0;
		for(Book book : books){
			totalPrice = totalPrice + book.getPrice();
		}
		return totalPrice;
	}

	public void addBook(Book book){
		try {
			bookRepository.save(book);
		} catch (SQLException e) {
			// log exception
			throw new DatabaseWriteException("Unable to write in database due to - " + e.getMessage());
		}
	}
}
```

4. Create DatabaseReadException

```
public class DatabaseReadException extends RuntimeException {
	public DatabaseReadException(String message) {
		super(message);
	}
}
```

5. Create DatabaseWriteException

```
public class DatabaseWriteException extends RuntimeException {
	public DatabaseWriteException(String message) {
		super(message);
	}
}

@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

	@InjectMocks
	BookService bookService;
	@Mock
	BookRepository bookRepo;

	@Test  //non void method
	public void testGetTotalPriceOfBooks() throws SQLException{
		//when(bookRepo.findAllBooks()).thenThrow(SQLException.class);
		 //when(bookRepo.findAllBooks()).thenThrow(new SQLException("Database is not found");
		//given(bookRepo.findAllBooks()).willThrow(SQLException.class);
		 assertThrows(DatabaseReadException.class,()->bookService.getTotalPriceOfBooks());
	}

	@Test  //void method
	public void testAddBook() throws SQLException{
		 Book b1=new Book("1234","Mockito",500,LocalDate.now());
		 //doThrow(SQLException.class).when(bookRepo).save(b1);
		 //doThrow(new SQLException("Database is not found")).when(bookRepo).save(b1);
		 assertThrows(DatabaseWriteException.class,()->bookService.addBook(b1));
	}
}
```

3. Behaviour verification
   - One of the primary benefit of mockito is that when mock object is created it remembers all operations performed on it
   - used to verify that mock method was called by system or not
   - using verify()

1. create model class

```
public class Book {
	private String bookId;
	private String title;
	private int price;
	private LocalDate publishedDate;
}

public class BookRequest {
	private String title;
	private int price;
	private LocalDate publishedDate;
}


2. create repo
public interface BookRepository {
	void save(Book book);
	Book findBookById(String bookId);
}

3. create service class

public class BookService {

	private BookRepository bookRepository;

	public BookService(BookRepository bookRepository) {
		this.bookRepository = bookRepository;
	}

	public void addBook(Book book) {
		if(book.getPrice() <= 500){
			return;
		}
		bookRepository.save(book);
	}

	public void updatePrice(String bookId, int updatedPrice){
		if(bookId==null) {
			return;
		}
		Book book = bookRepository.findBookById(bookId);
		book.setPrice(updatedPrice);
		bookRepository.save(book);
	}
}

@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

	@InjectMocks
	BookService bookService;
	@Mock
	BookRepository bookRepo;

	//To verify bookRepo is invoked or not, since it is an mock object
	@Test
	public void testAddBook() {
		Book b1=new Book("1234","Mockito",600,LocalDate.now());
		bookService.addBook(b1);
		verify(bookRepo).save(b1);
	}

	//To verify number of invocations of  mock method, if book price is <= 500 then it wont invoke save()
	@Test
	public void testAddBook1()  {
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
		bookService.addBook(b1);
		verify(bookRepo,times(0)).save(b1);
	}
```

```
	@Test
	public void testAddBook2()  {
		Book b1=new Book("1234","Mockito",600,LocalDate.now());
		Book b2=new Book("1235","Mockito",700,LocalDate.now());
		bookService.addBook(b1);
		bookService.addBook(b2);
		//verify(bookRepo,times(2)).save(b1);
		//verify(bookRepo,atLeast(4)).save(b1);
		//verify(bookRepo,atMost(2)).save(b1);
		//verify(bookRepo,atMostOnce()).save(b1);
		verify(bookRepo,atLeastOnce()).save(b1);
	}

	//Here price is 500, so it wont invoke save(), previously we checked using times(0)
	//instead we can use never()
	@Test
	public void testAddBook3()  {
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
		bookService.addBook(b1);
		verify(bookRepo,never()).save(b1);
	}

	@Test
	public void testUpdatePrice() {
		Book b1=new Book("1234","Mockito",700,LocalDate.now());
		bookService.updatePrice("1234", 200);
		verify(bookRepo).save(b1);
	}

	//Verify no interaction with mock object
	@Test
	public void testUpdatePrice1() {
		bookService.updatePrice(null, 200);
		verifyNoInteractions(bookRepo);
	}

	//verify no unexpected interaction
	@Test
	public void testUpdatePrice2() {
		Book b1=new Book("1234","Mockito",700,LocalDate.now());
		when(bookRepo.findBookById("1234")).thenReturn(b1);
		bookService.updatePrice("1234",700);
		verifyNoMoreInteractions(bookRepo);
	}

	//To call mock object in order, if we change the order the test case failed
	@Test
	public void testUpdatePrice3() {
		Book b1=new Book("1234","Mockito",700,LocalDate.now());
		when(bookRepo.findBookById("1234")).thenReturn(b1);
		bookService.updatePrice("1234",800);
		InOrder o=Mockito.inOrder(bookRepo);
		o.verify(bookRepo).findBookById("1234");
		o.verify(bookRepo).save(b1);
	}
}
```

4. ArgumentCaptor
   - we can capture the arguments passed to mock methods

1. Create model class

```
public class Book {
	private String bookId;
	private String title;
	private int price;
	private LocalDate publishedDate;
}

public class BookRequest {
	private String title;
	private int price;
```

```
	private LocalDate publishedDate;
}

2. Create repo
public interface BookRepository {
	void save(Book book);
}

3. Create service class

public class BookService {

	private BookRepository bookRepository;

	public BookService(BookRepository bookRepository) {
		this.bookRepository = bookRepository;
	}


	public void addBook(BookRequest bookRequest) {
		Book book = new Book();
		book.setTitle(bookRequest.getTitle());
		book.setPrice(bookRequest.getPrice());
		book.setPublishedDate(bookRequest.getPublishedDate());
		bookRepository.save(book);
	}

}

@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

	@InjectMocks
	BookService bookService;
	@Mock
	BookRepository bookRepo;
	@Captor
	ArgumentCaptor<Book> bookCaptor;

	@Test
	public void testAddBook() {
		BookRequest req=new BookRequest("Mockito",500,LocalDate.now());
		bookService.addBook(req);
		verify(bookRepo).save(bookCaptor.capture());
		Book book=bookCaptor.getValue();
		assertEquals("Mockito",book.getTitle());
	}

}

5. ArgumentMatchers
    - used to pass some dynamice values to the arguments, it is like wildcard where instead of specific input we can specify the range or type of input
    - ArgumentMatchers should be provide for all arguments, either u provide all arg with argument matcher or provide fixed value, dont use
combination, then testcase failed
    - Argumentmatchers can be used only with when() or with verify(), u cant call argument matches while calling the method
    - Specific type of argument matchers - anyBoolean(), anyByte(), anyChar(), anyInt(), anyFloat(), anyDouble(), anyLong(), anyShort(), anyString()
    - Collection type argument matchers - anyList(), anySet(), anyMap()
    - String type argument matcher - matches(string), startsWith(string), contains(string), endsWith(String)

3 methods
1. any() - any object or null
  any(Class c)
  anyVararg() - any vararg

1. Create model class
public class Book {
	private String bookId;
	private String title;
	private int price;
	private LocalDate publishedDate;
	private boolean isDigital;
```

```
}

2. Create repo
public interface BookRepository {
	void save(Book book);
	Book findBookById(String bookId);
	Book findBookByTitleAndPublishedDate(String title, LocalDate localDate);
	Book findBookByTitleAndPriceAndIsDigital(String title, int price, boolean isDigital);
	void saveAll(List<Book> books);
}

3. Create service class

public class BookService {

	private BookRepository bookRepository;

	public BookService(BookRepository bookRepository) {
		this.bookRepository = bookRepository;
	}


	public void updatePrice(String bookId, int updatedPrice){
		Book book = bookRepository.findBookById(bookId);
		book.setPrice(updatedPrice);
		bookRepository.save(book);
	}

	public Book getBookByTitleAndPublishedDate(String title, LocalDate localDate) {
		return bookRepository.findBookByTitleAndPublishedDate(title, localDate);
	}

	public Book getBookByTitleAndPriceAndIsDigital(String title, int price, boolean isDigital) {
		return bookRepository.findBookByTitleAndPriceAndIsDigital(title, price, isDigital);
	}

	public void addBooks(List<Book> books) {
		bookRepository.saveAll(books);
	}
}

@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

	@InjectMocks
	BookService bookService;
	@Mock
	BookRepository bookRepo;

	@Test
	public void testUpdatePrice() {
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
		//when(bookRepo.findBookById(any())).thenReturn(b1);
		when(bookRepo.findBookById(any(String.class))).thenReturn(b1);
		bookService.updatePrice("1234",200);
		verify(bookRepo).save(b1);
	}

	//Invalid use of Argument	Matchers
	@Test
	public void testInvalidArgumentMatchers() {
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
	//when(bookRepo.findBookByTitleAndPublishedDate("Mockito",any())).thenReturn(b1);  //wrong
	   Book b2=bookService.getBookByTitleAndPublishedDate("Mockito",any());  //wrong
	}

	@Test
	public void testSpecificMatchers() {
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
		when(bookRepo.findBookByTitleAndPriceAndIsDigital(anyString(), anyInt(), anyBoolean())).thenReturn(b1);
```

```
		Book b2=bookService.getBookByTitleAndPriceAndIsDigital("Mockito",600, false);
		assertEquals("Mockito",b2.getTitle());
	}

	@Test
	public  void testCollectionMatchers() {
		List<Book> books=new ArrayList<>();
		Book b1=new Book("1234","Mockito",500,LocalDate.now());
		books.add(b1);
		bookService.addBooks(books);
		verify(bookRepo).saveAll(anyList());
	}

	@Test
	public void testStringMatchers() {
		Book b1=new Book("1234","Mockito in action",500,LocalDate.now());
	  when(bookRepo.findBookByTitleAndPriceAndIsDigital(contains("action"), anyInt(), anyBoolean())).thenReturn(b1);
		Book b2=bookService.getBookByTitleAndPriceAndIsDigital("Mockito in action", 700, true);
		assertEquals("Mockito in action",b2.getTitle());
	}
}
```

Subject: RE: Reg: Day 23 Notes

Classification: **Confidential**

SDLC
1. Estimation and Planning
2. Requirement Analysis - what actually we do in the project?
3. Design - How we develop the project? - 40%
4. Coding - 20%
5. Testing - 40%
6. Implementation
7. Maintainence

Unit testing - developer will do test testing - test individual methods in classes

Junit framework - we can perform unit testing in Java

Why Junit?
- method by method we are going to test
- In real life cases it depends whether we are using junit or mockito

JUnit 4 - junit4.13+mockito-core 3.2.4 - min jdk1.5+single jar file
JUnit 5 - latest version - Jupiter - 5th planet in solar system
junit5 = junit jupiter+mockito junit jupiter - min JDK1.8 + 3 Jar files = junit platform + junit jupiter + junit vintage

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

import com.pack.ExampleUtil;

public class TestExampleUtil {
   @Test
	public void testAdd() {
		ExampleUtil e=new ExampleUtil();
		assertEquals(5, e.add(1, 4));
		assertTrue("hello".length()==5);
		assertFalse("hello".length()==4);
		String s1=null;
		String s2="abc";
		assertNull(s1);
		assertNotNull(s2);
```

```
	}
}
```

Lifecycle methods
1. @BeforeAll - the methods with @BeforeAll will be invoked first before all testcases are executed
2. @BeforeEach - the methods with @BeforeEach will be invoked before every testcases are executed
3. @AfterAll - the methods with @AfterAll will be invoked first after all testcases are completed
4. @AfterEach - the methods with @AfterEach will be invoked after every testcases are completed

```
public class TestExampleUtilLifeCycle {

	@BeforeAll
	public static void beforeAll() {
		System.out.println("Executed before all testcases");
	}

	@BeforeEach
	public void beforeEach() {
		System.out.println("Executed before every testcases");
	}

	@AfterAll
	public static void afterAll() {
		System.out.println("Executed after all testcases");
	}

	@AfterEach
	public void afterEach() {
		System.out.println("Executed after every testcases");
	}

	@Test
	public void test1() {
		System.out.println("Inside test1");
	}
	@Test
	public void test2() {
		System.out.println("Inside test2");
	}
	@Test
	public void test3() {
		System.out.println("Inside test3");
	}

}
```

Junit 4	Junit5
1. @BeforeClass	1. @BeforeAll
2. @Before	2. @BeforeEach
3. @AfterClass	3. @AfterAll
4. @After	4. @AfterEach
5. @Ignore	5. @Disabled

```
public class TestExampleUtilLifeCycle {

	@BeforeAll
	public static void beforeAll() {
		System.out.println("Executed before all testcases");
	}

	@BeforeEach
	public void beforeEach() {
		System.out.println("Executed before every testcases");
	}

	@AfterAll
	public static void afterAll() {
		System.out.println("Executed after all testcases");
	}
```

```
	@AfterEach
	public void afterEach() {
		System.out.println("Executed after every testcases");
	}

	@Test
	public void test1() {
		System.out.println("Inside test1");
	}
	@Test
	@Disabled("Ignoring test")
	public void test2() {
		System.out.println("Inside test2");
	}
	@Test
	@DisplayName("Third testcase")
	public void test3() {
		System.out.println("Inside test3");
	}

}
```

Mockito framework
- open source testing framework to create mock objects
- Every appl we create we have 3 tier architecture (ie) controller, service and repository. If we write test case for controller, first it will hits controller - service - repository - db, but it is not good practice to hit db at time of testing
- If we want to mock repository data, whatever request comes from controller, it forwards to the service and instead of forwarding to actual repo and then to db, we create mock object of repo which acts as a database

---

Subject: RE: Reg: Day 23 Notes

Classification: **Confidential**

Different ways to communicate with database
1. Predefined methods
- T save(T t)- store single object
- T saveAll(Iterable) - store multiple object
- Optional findById(int id) - return single object
- Iterable findAll() - fetch all object
- boolean existsById(int id)
- void deleteById(int id) - delete single object based on id
- void delete(T t) - delete single object
- void deleteAll()

2. Using custom JPA method
- derived methods used for fetching the data based on other properties
- name of the method should start with findBy/readBy/getBy/queryBy
- declare all custom JPA method in proper format inside repository interface

like - pattern matching (%,_)

```
List<Employee> findByDept(String dname);
List<Employee> readByDeptAndSalaryLessThan(String dname,double sal);
List<Employee> getByNameLike(String pattern);
List<Employee> queryByNameLikeAndSalaryGreaterThan(String name,double sal);
List<Employee> findByDeptIsNull();
List<Employee> findByNameStartsWith(String name);
List<Employee> findByNameContainingOrDeptContainingAllIgnoreCase(String name, String dname);
```

3. Limiting the records based on custom JPA method
- using first or top keyword

```
Employee findFirstByOrderBySalaryDesc();
Employee findTopByOrderBySalaryAsc();
```

```
  List<Employee> findFirst3ByOrderBySalaryDesc();
  List<Employee> findTop5ByOrderBySalaryAsc();
  List<Employee> findFirst3ByDeptOrderBySalaryDesc(String dname);

4. count the elements based on custom JPA methods
  - using countBy

  long countByDept(String dname);
  long countByNameEndingWith(String name);
  long countBySalaryGreaterThanEqual(double sal);

5. If we want to perform joins or subqueries then we cant use custom JPA method, in that case we have to write the queries using @Query
 - 2 types of query
1. JPAQL - @Query - query the entity class
2. SQL - @Query - query the tables

@Query("select e from Employee e")  //JPAQL
  List<Employee> fetchAllEmployee();

  @Query(value="select * from empl2023", nativeQuery=true) //SQL
  List<Employee> fetchAllEmployee1();

6. Passing parameters to the query - 2 ways

1. Positional parameter - using ? - ?1,?2,?3....

@Query("select e from Employee e where e.name like ?1 and e.salary=?2")
  List<Employee> findEmpByNameAndSalary(String name, Double sal);

2. Named parameter - using : - :a,:abc,:xyz

@Param - used to assign value to named parameter

@Query("select e from Employee e where e.name like :a and e.salary=:xy")
  List<Employee> findEmpByNameAndSalary1(@Param("a")String name, @Param("xy")Double sal);

7. If we want to perform DML operation(insert,update,delete) using @Query, we have to provide 2 more annotation like @Modifying and
@Transactional

  @Modifying
  @Transactional
  @Query("update Employee e set e.salary=e.salary+e.salary*:percent/100 where e.dept=:dname")
  int updateSalary(@Param("dname")String dname, @Param("percent")double percentage);

8. Instead of writing queries in repository interface, we can write queries in entity class and we can identify those queries by their name
 JPAQL - @NamedQuery
 SQL - @NamedNativeQuery

@Entity
@Table(name="empl2023")
@Data
@NoArgsConstructor
@AllArgsConstructor
@NamedQuery(name="Employee.fetchEmpBySalary",query="select e from Employee e where e.salary=?1")
@NamedNativeQuery(name="Employee.fetchEmpBySalary1",query="select * from empl2023 where e.salary=:salary",resultClass=Employee.class)
public class Employee {
 @Id
 private Integer id;
 private String name;
 private String gender;
 private String email;
 private String dept;
 private Double salary;
}

List<Employee> fetchEmpBySalary(double salary);
List<Employee> fetchEmpBySalary1(@Param("salary")double sal);

9. For Sorting we have Sort class

List<Employee> findByDeptOrderBySalaryDesc(String dept);
```

```
List<Employee> findByDept(String dname, Sort s);

List<Employee> l=empRepo.findByDept("Sales", Sort.by("salary").descending());
l.forEach(System.out::println);
List<Employee> l=empRepo.findByDept("Sales", Sort.by("salary").ascending());
l.forEach(System.out::println);
```

10. Consider we have entity class with 30 properties, but we need to display only 10 properties
Consider we have employee table, now we want to display total number of male and female employees, so the output would be

```
mysql> select count(gender) as "Total count", gender from empl2023 group by gender;
+-------------+--------+
| Total count | gender |
+-------------+--------+
|           5 | male   |
|           2 | female |
+-------------+--------+
2 rows in set (0.01 sec)

@Query("select count(gender) as "Total count", gender from empl2023 group by gender");
List<Employee> countGenderWise(); -- wrong - it will display all properties


@Query("select count(gender) as "Total count", gender from empl2023 group by gender");
List<Object[]> countGenderWise(); - yes, but it is not a good practice to return an object

@Query("select count(gender) as "Total count", gender from empl2023 group by gender");
Map<Integer,String> countGenderWise(); - wrong, because spring data jpa dosent support Map as return type

Constructor Method - used when we want to display only specific properties

@Data
@AllArgsConstructor
@NoArgsConstructor
public class GenderCount {
    private Long count;
    private String gender;
}

@Query("select new com.pack.SpringData.GenderCount(count(e.gender),e.gender) from Employee e group by e.gender")
  List<GenderCount> countGenderWise();
```

Subject: RE: Reg: Day 22 Notes

Classification: **Confidential**

Handson
1. Create springboot appl to insert student details into the db using commandline runner

@Value
1. Injecting properties one by one
2. Loose binding/loose grammar is not supported (ie) attribute name should be matching
3. Support SpEL (${})
4. Validation of properties is not supported
4. Support only scalar datatype

@ConfigurationProperties
1. Bulk injection of properties
2. Loose binding/loose grammar is supported (ie) no need to match attribute name (ie) special char or cases
3. Not support SpEL
4. Validation of properties is supported
5. Supports all datatypes as well as objects

mail.properties
#scalar datatype

```
mail.to=abc@gmail.com
mail.from=xyz@gamil.com
mail.age=25
mail.firstname=Ramu
mail.lastname=kumar

#complex datatype
mail.cc=uvw@gmail.com,efh@gmail.com
mail.bcc=mno@gmail.com,pqr@gmail.com

#Nested datatype
mail.credential.username=ram
mail.credential.password=abcde
```

To do validation

```
<dependency>
                        <groupId>org.hibernate.validator</groupId>
                        <artifactId>hibernate-validator</artifactId>
                        <version>6.0.5.Final</version>
                </dependency>
                <dependency>
                        <groupId>javax.validation</groupId>
                        <artifactId>validation-api</artifactId>
                        <version>2.0.0.Final</version>
                </dependency>
```

@Validated - to do validating the properties in properties file
@Valid - to do validation for nested class properties

```
@Configuration
@PropertySource("classpath:mail.properties")
@ConfigurationProperties(prefix="mail")
@Data
@Validated
public class MailProps {
        @NotBlank
   private String to;
        @NotBlank
   private String from;
        @Min(value=20)
        @Max(value=40)
   private Integer age;
        private String FIRSTNAME;

   private String[] cc;
   private List<String> bcc;

   @Valid
   private Credential credential=new Credential();

   @Data
   public class Credential {
        @NotBlank
        private String username;
        @Size(max=8,min=4)
        private String password;
   }
}


@RestController
@Log4j2
public class StudentController {

        @Value("${student.id}")
        private Integer id;

        @Value("${student.mark}")
        private Integer mark;
```

```
	@Value("${mail.lastname}")
	private String lname;

	@Autowired
	StudentProps stuProp;

	@Autowired
	MailProps mail;

	@GetMapping("/student")
	public String getStudentInfo() {
		return id+" "+mark+" "+stuProp.getFirstname()+" "+stuProp.getAddress();
	}

	@GetMapping("/mail")
	public String mailInfo() {
		return mail.getFrom()+" "+mail.getCc()+" "+mail.getBcc()+" "+mail.getCredential().getUsername()+" "+mail.getFIRSTNAME();
	}
}
```

Springboot Interceptors
- used to intercept client request and response
- Interceptors are similar to Filters(servlet), but interceptors are applied to the request that are sending to the controller prg
- We have to implement HandlerInterceptor interface - 3 methods

1. preHandle() - perform any operation before sending request to controller
2. postHandle() - perform any operation before sending response to client
3. afterCompletion() - perform any operation after completing request and response

```
@RestController
@Log4j2
public class EmployeeController {

	@GetMapping("/emp")
	public String getEmployeeInfo() {
		log.info("Inside EmployeeController");
		return "Employee are working";
	}
}

@Component
@Log4j2
public class TimerInterceptor implements HandlerInterceptor {

   @Override
	public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
			throws Exception {
		log.info("Inside preHandle");
		request.setAttribute("startTime", System.currentTimeMillis());
		return true;
	}

   @Override
   public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
		ModelAndView modelAndView) throws Exception {
	log.info("Inside postHandle");
   }

   @Override
   public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)
		throws Exception {
	log.info("Inside afterCompletion");
	long totalTime=System.currentTimeMillis()-(long)request.getAttribute("startTime");
	System.out.println("Total time taken is "+totalTime);
   }
}


@Configuration
public class EmployeeConfig implements WebMvcConfigurer {
```

```
        @Autowired
        TimerInterceptor timerInterceptor;

        @Override
        public void addInterceptors(InterceptorRegistry registry) {
           //registry.addInterceptor(timerInterceptor); //this interceptor will be invoked for all controller prg
        registry.addInterceptor(timerInterceptor).addPathPatterns("/emp","/mail");
        }
}
```

JDBC
JPA

Spring Data JPA
- Used to persist data into database
- It is a library that adds an extra layer of abstraction on top of JPA providers

JPA providers - vendors that provide the implementation of JPA specification like Hibernate, iBatis, Toplink etc

JPA specification - provides mapping of entity class with column of database table using @Entity, @Table, @Id etc

3 layers
1. Spring Data JPA - Create JPA Repository - 2 interface
a. JpaRepository<Entityclassname, datatype of PK> interface - used to perform CRUD and batch operation
- T getById(int id) - deprecated
- T getOne(int id) - deprecated
- T getReferenceById(int id) - fetch single object
- List<T> findAll() - return multiple object
- T saveAndFlush(T t) - store single object into db
- T saveAllAndFlush(Iterable) - store multiple object into db
- void deleteAllByIdInBatch(Iterable)
- void deleteAllInBatch()

b. JpaSpecificationExecutor<Entityclassname> interface - used to retrieve data
based on some condition

2. Spring data commons layer - 3 interface
a. Repository<Entityclassname, datatype of PK> interface - marker interface
b. CrudRepository<Entityclassname, datatype of PK> interface - used to perform Crud operation
- T save(T t)- store single object
- T saveAll(Iterable) - store multiple object
- Optional findById(int id) - return single object
- Iterable findAll() - fetch all object
- boolean existsById(int id)
- void deleteById(int id) - delete single object based on id
- void delete(T t) - delete single object
- void deleteAll()

c. PagingAndSortingRepsitory<Entityclassname, datatype of PK> interface - used for paging and sorting purpose

3. JPA providers - vendors that provide the implementation of JPA specification like Hibernate, iBatis, Toplink etc

Repository interface
  extends
CrudRepository interface
  extends
PagingAndSortingRepository interface
  extends
JpaRepository interface

1. Create spring boot project with spring data jpa, mysql, lombok dependency

2. Configure db info in application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/jpa
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
#dialect will generate the query based on particular db
spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
```

3. Create entity class

```
@Entity
@Table(name="empl2023")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
 @Id
 private Integer id;
 private String name;
 private String gender;
 private String email;
 private String dept;
 private Double salary;
}
```

4. Create repository interface

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

5.

```
@SpringBootApplication
public class SpringDataApplication implements CommandLineRunner{

	@Autowired
	EmployeeRepository empRepo;

	public static void main(String[] args) {
		SpringApplication.run(SpringDataApplication.class, args);
	}

	@Override
	public void run(String... args) throws Exception {
		//insertEmployee();
		//fetchEmployee(104);
		//fetchAllEmployee();
		//updateEmployee(103);
		deleteEmployee(103);
	}

	private void deleteEmployee(int i) {
		if(empRepo.existsById(i)) {
			//Employee e=empRepo.findById(i).get();
			//empRepo.delete(e);
			empRepo.deleteById(i);
		}
	}

	private void updateEmployee(int i) {
		if(empRepo.existsById(i)) {
			Employee e=empRepo.findById(i).get();
			e.setSalary(60000.0);
			empRepo.save(e);
		}
	}

	private void fetchAllEmployee() {
		List<Employee> l=empRepo.findAll();
		l.forEach(System.out::println);
	}

	private void fetchEmployee(int i) {
		Optional opt=empRepo.findById(i);
		System.out.println(opt.get());
	}

	private void insertEmployee() {
		/*Employee e1=new Employee(100,"Ram","male",ram@gmail.com,"HR",20000.0);
```

```
		empRepo.save(e1);*/

		List<Employee> l1=new ArrayList<>();
		Employee e1=new Employee(101,"Sam","male",sam@gmail.com,"Sales",25000.0);
		l1.add(e1);
		Employee e2=new Employee(102,"Saj","male",saj@gmail.com,"IT",35000.0);
		l1.add(e2);
		Employee e3=new Employee(103,"Lim","female",lim@gmail.com,"HR",15000.0);
		l1.add(e3);
		Employee e4=new Employee(104,"Amy","female",amy@gmail.com,"Sales",36000.0);
		l1.add(e4);
		Employee e5=new Employee(105,"Adam","male",adam@gmail.com,"IT",45000.0);
		l1.add(e5);
		Employee e6=new Employee(106,"Tam","male",Tam@gmail.com,"Sales",55000.0);
		l1.add(e6);
		empRepo.saveAll(l1);
	}

}
```

Subject: RE: Reg: Day 21 Notes

Classification: **Confidential**

Handson
1. Create Spring boot application to implement profiles likes prod,dev and print value of message from properties file.
Note: create all profiles in single file

2. Create Spring boot application to read and print all user information from user.properties file present in local machine(ie)C:/handson folder

3. Create Spring Boot MVC appl, to display employee details like name, age, salary in employee.jsp

Spring MVC - Where we return response as JSP page

client request - Controller(used to handle req and res) - Service (write logic) - model class - view page(JSP)

1. Create Spring boot project with spring web dependency and with war packaging

To make Springboot understand JSP, we have provide 2 dependency

```
	<dependency>
		<groupId>org.apache.tomcat.embed</groupId>
		<artifactId>tomcat-embed-jasper</artifactId>
		<scope>provided</scope>
	</dependency>
	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>jstl</artifactId>
	</dependency>
```

2. Create Controller prg

```
@Controller
public class MainController {

	@GetMapping("/info")
	public String getInfo() {
		return "index";  //name of jsp page
	}

	@GetMapping("/info1")
	public String getInfo1(ModelMap m,Model m1,Map m2) {
		m.addAttribute("name", "Ram");
		m1.addAttribute("age", 24);
		m2.put("mark", 80);
		return "first";
	}

}
```

3. Create jsp page as per return value inside webapp - WEB-INF - views(any name) - create jsp page

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Welcome to SPRING MVC</h2>
</body>
</html>
```

4. Configure jsp info in application.properties

```
server.port=1000
server.servlet.context-path=/mvc
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
#prefix+viewname+suffix=/WEB-INF/views/index.jsp
```

To pass value from controller prg to JSP prg we can use ModelMap class or Model class or Map intf

5. Create first.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Welcome ${name} has ${age} and with ${mark}</h2>
</body>
</html>
```

Springboot Jetty
1. Exclude Tomcat server from web dependency

```
<dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                        <exclusions>
                          <exclusion>
                            <groupId>org.springframework.boot</groupId>
                            <artifactId>spring-boot-starter-tomcat</artifactId>
                          </exclusion>
                        </exclusions>
                </dependency>
```

2. Comment ur tomcat server and tomcat-embed-jasper, jstl
3. Include Jetty dependency

```
<dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jetty</artifactId>
                </dependency>
```

4. To make Jetty to understand JSP page

```
        <dependency>
                        <groupId>org.eclipse.jetty</groupId>
                        <artifactId>apache-jsp</artifactId>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.eclipse.jetty</groupId>
                        <artifactId>apache-jstl</artifactId>
                </dependency>
```

CommandLineRunner interface
- We can execute any task just before spring boot application startup
- public void run(String...s){
}

```
@SpringBootApplication
@Order(value=2)
public class SpringBoot2Application implements CommandLineRunner{

	protected final static Log log=LogFactory.getLog(SpringBoot2Application.class);

	public static void main(String[] args) {
		SpringApplication.run(SpringBoot2Application.class, args);
		log.info("Inside main method");
	}

	@Override
	public void run(String... args) throws Exception {
		log.info("inside run method");
	}

}

@Component
@Order(value=3)
public class CommandLineRunner1 implements CommandLineRunner {

	protected final static Log log=LogFactory.getLog(CommandLineRunner1.class);

	@Override
	public void run(String... args) throws Exception {
		log.info("Inside commandlinerunner1");
	}

}

@Component
@Order(value=1)
public class CommandLineRunner2 implements CommandLineRunner {

	protected final static Log log=LogFactory.getLog(CommandLineRunner2.class);

	@Override
	public void run(String... args) throws Exception {
		log.info("Inside commandlinerunner2");
	}

}
```

ApplicationRunner interface
- We can execute any task just before spring boot application startup
- public void run(ApplicationArgument s){
}

Spring boot profiling
- In enterprise appl we have many env like dev, test, prod etc and each env needs specific configuration related to that env and configured in application.properties, we cant configure everything in single file so to use different properties file in different env we use profiling

1. we need to create different env related properties files by application-profilename.properties (eg) application-dev.properties, application-prod.properties
2. spring.profiles.active=profilename in application.properties, it will invoke the related profiles

@Value - used to read property from application.properties to controller prg

@Profile - used to programmatically control files based on profiles

application.properties
```
message=Welcome default user
server.port=2000
spring.profiles.active=dev
```

application-prod.properties
message=Welcome production user
server.port=1002

application-dev.properties
message=Welcome development user
server.port=1001

```
@RestController
public class ProfileController {

	@Value("${message}")  //Spring Expression Language
	private String msg;

	@GetMapping("/")
	public String hello() {
		return "Hello world from controller " + msg;
	}
}


@Configuration
@Profile("prod")
public class AppConfig {
   @PostConstruct
	public void print() {
		System.out.println("This method is invoked only for production profile");
	}
}
```

application.properties
1. It is represent as a sequence of key value pair

```
server.port=1000
server.servlet.context-path=/app
```

2. This file is supported only in Java lang

3. support only key value pairs in the form of string

4. If we want to handle different profiles then we have to create different properties files

application.yml
1. It is represent in hiearchial format

```
server:
  port: 1000
  servlet:
    context-path: /app

spring:
  profiles:
    active: dev
```

2. This file is supported in Java, Python etc

3. support key value pairs, map, list, scalar datatype

4. All configuration related to multiple profiles can be handled in single yml file

```
server:
  port: 1000
  servlet:
    context-path: /app

spring:
  profiles:
    active: dev

---
```

```
spring:
  profiles: dev
server:
  port: 2000

---
spring:
  profiles: prod
server:
  port: 2001
```

@PropertySource - used to read single property file with different name present in different location
@PropertySources - used to read multiple property files

@ConfigurationProperties - used to map entire properties in properties file to a separate java bean object

Lombok dependency - used to reduce boilerplate code (ie) getters, setters, default constructor, argument constructor, toString, log

```
student.properties
student.id=1000
student.name=Ram
student.address=Chennai
student.age=24

student1.properties
student.email=ram@gmail.com
student.course=CSE
student.mark=89

@Configuration
//@PropertySource("classpath:student.properties")
//@PropertySource("file:\\C:\\Training\\student1.properties")
@PropertySources({
	@PropertySource("classpath:student.properties"),
	@PropertySource(file:\\C:\\Training\\student1.properties)
})
@ConfigurationProperties(prefix="student")
//@Getter
//@Setter
//@NoArgsConstructor
//@AllArgsConstructor
//@ToString
@Data
public class StudentProps {
  private String name;
  private String address;
  private Integer age;
  private String email;
  private String course;
}


@RestController
@Log4j2
public class StudentController {

	@Value("${student.id}")
	private Integer id;

	@Value("${student.mark}")
	private Integer mark;

	@Autowired
	StudentProps stuProp;

	@GetMapping("/student")
	public String getStudentInfo() {
		return id+" "+mark+" "+stuProp.getName()+" "+stuProp.getAddress();
	}
}
```

Subject: RE: Reg: Day 20 Notes

Classification: **Confidential**

JPA QL - Directly query the entity class

NativeQuery - query the table directly(SQL)

```
@Entity
@Table(name="person100")
@NamedQueries({  //identify JPAQL query by their name
	@NamedQuery(name="findPerson", query="select p from Person p"),
	@NamedQuery(name="findPersonById", query="select p.name from Person p where p.id=:personid")
})
@NamedNativeQueries({  //identify sql query by their name
	@NamedNativeQuery(name="findAllPerson", query="select * from person100", resultClass = Person.class)
})
public class Person {
  @Id
  private Integer id;
  private String name;
  private String address;
  private Integer age;
  private String city;
}

public class Main2 {

	public static void main(String[] args) {
		EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
		EntityManager em=emf.createEntityManager();
		EntityTransaction et=em.getTransaction();
		et.begin();

		//9. Native Query - query the table
		/*Query q=em.createNativeQuery("select * from person100");
		List l=q.getResultList();
		Iterator i=l.iterator();
		while(i.hasNext()) {
			Object[] o=(Object[])i.next();
			System.out.println(o[0]+" "+o[1]+" "+o[2]);
		}*/

		/*Query q=em.createNativeQuery("select * from person100",Person.class);
		List<Person> l=q.getResultList()
		for(Person p:l)
			System.out.println(p.getAddress());*/

		//10. Named JPAQL query
		/*Query q=em.createNamedQuery("findPerson");
		List<Person> l=q.getResultList();
		for(Person p:l)
			System.out.println(p.getAddress());*/

		/*Query q=em.createNamedQuery("findPersonById");
		q.setParameter("personid", 100);
		String s=(String)q.getSingleResult();
		System.out.println(s);*/

		Query q=em.createNativeQuery("findAllPerson");
		List<Person> l=q.getResultList();
		for(Person p:l)
			System.out.println(p.getAddress());


		et.commit();
		System.out.println("Success");
```

```
                em.close();
                emf.close();

        }

}
```

Logging framework - log4j,slf4j

Spring boot framework
- open source framework, used to create faster java based application/web based appl, it is not a replacement of spring framework instesd it provides production ready appl, so as a developer we are focusing mainly on business logic rather that doing all external configuration and downloading the jar files
- Developed Pivotal team

Advantage
1. easy to understand, develop faster appl so increase productivity
2. reduce ur development time
3. Avoid xml configurations
4. Everything is autoconfigured, so no need for manual configuration
5. Springboot comes with embedded server like Apache Tomcat(Default), Jetty, Undertow
6. Springboot comes with in-memory database called h2 database used for testing purpose
7. Springboot comes with inbuilt logging framework called logback,slf4j

Drawbacks
1. Migration effort - converting from existing spring project to springboot is not straight forward
2. Deploying spring boot appl on other servers like jboss, weblogic etc is not straight forward
3. Developed spring boot keeping microservice and cloud in mind

Spring
1. used to develop JavaEE framework for building appl(client + server)
2. main feature is DI
3. develop loosely coupled appl
4. write lot of bailerplate code
5. we have to explicitly configure the server
6. dosent have inmemory db
7. manually define dependencies in pom.xml
8. requires bean configuration in separate xml file

Springboot
1. used to develop REST API(server part) based on MVC architecture
2. main feature is autoconfiguration
3. create standalone appl with less configuration
4. reduce boilerplate code
5. Springboot comes with embedded server
6. comes with inmemory db
7. Spring boot comes spring boot starter parent which will internally download all necessary jar files
8. No xml configuration is needed

Spring boot dependency management
- manages dependencies and configuration automatically
- Spring-boot-starter-parent is a project starter, provide all basic configuration(jar files) to develop spring boot appl
- avoid version conflicts
- dependency section tells maven what jars to be downloaded, parent section tells what version of jars to be downloaded

```
    <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.7.17</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
```

Developing Spring boot appl - 3 ways
1. Using Spring initializer - https://start.spring.io/
2. Using STS(Spring Tool Suite) - IDE for springboot
3. Springboot CLI(Command Line Interface)

Springboot annotation - 3 types
1. Spring core annotation
   1. @Configuration
   2. @Bean

3. @Autowired
4. @Qualifier
5. @Required
6. @Component
7. @Service, @Controller, @Repository

2. Spring MVC annotation
- Controller prg - when client gives the request it will come to controller prg, for that particular request we have to write business logic

@Controller - return the response in view page(JSP)
@RestController - @Controller + @ResponseBody - return the return in json format

@RequestMapping - used to map the request for a logic
@RequestMapping(value="/fetchEmployee",method=RequestMethod.GET) - used to give GET request
@RequestMapping(value="/insertEmployee",method=RequestMethod.POST) - used to give POST request
@RequestMapping(value="/updateEmployee",method=RequestMethod.PUT) - used to give PUT request
@RequestMapping(value="/removeEmployee",method=RequestMethod.DELETE) - used to give DELETE request
- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

@RequestBody - used to get input as json format
@RequestParam - used to return value present between ? and &
@PathVariable - used to get value from the request

3. Spring boot annotation
- @SpringBootApplication - @EnableAutoConfiguration + @Configuration + @ComponentScan

1. Create Springboot project from Spring Initialzr

https://start.spring.io/

Project: Maven
Language: Java
Springboot version: 2.7.17

Group: com.pack
Artifact: SpringBoot1 (project name)
Packaging: jar
Java: 11
Click Add Dependencies

Spring web dependency - Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Click Generate

2. Right click project - click Extract all - Click Browse - Choose eclipse workspace - click Select Folder - Click Extract

3. Goto File - Import - Select Existing Maven project - Click Next - Click Browse - Select extracted project from workspace - Click Select folder - click Finish

4. Right click project - select properties - Under Location - copy project path

5. Open command prompt
>cd "paste project path"
>mvn clean install

6. Right click project - Maven - Update project - Force update - click ok

7. Create controller prg

```
@RestController
public class ExampleController {

	@RequestMapping(value="/msg1",method=RequestMethod.GET)
	public String getMessage() {
		return "Welcome to Spring Boot";
	}

	@GetMapping(value="/msg2")
	public String getMessage1() {
		return "Hello to Spring Boot";
	}
```

```
}
```

By default, Spring boot will read all configuration from application.properties/application.yml present in src/main/resources

```
server.port=2000
```

8. Start the appl
http://localhost:2000/msg1
http://localhost:2000/msg2

9. We have to provide context path, in application.properties
server.servlet.context-path=/app

http://localhost:2000/app/msg1
http://localhost:2000/app/msg2

10. If we provide wrong request it will give whitelabel error page, instead we want to print some error message for that we have to implement ErrorController interface and override getErrorPath()

```
@RestController
public class ExampleController implements ErrorController{

	private static final String PATH="/error";

	@RequestMapping(value="/msg1",method=RequestMethod.GET)
	public String getMessage() {
		return "Welcome to Spring Boot";
	}

	@GetMapping(value="/msg2")
	public String getMessage1() {
		return "Hello to Spring Boot";
	}

	@GetMapping(value=PATH)
	public String getErrorMsg() {
		return "Requested Page does not found";
	}

	public String getErrorPath() {
		return PATH;
	}

}
```

11. When we create any program it should be always present as subpackage of main class, if we create outside the main class pkg then we have to use @ComponentScan which is used to scan the package and execute the prg

```
@SpringBootApplication
@ComponentScan(basePackages = {"com.hcl","com.pack.SpringBoot11"})
public class SpringBoot11Application {

	public static void main(String[] args) {
		SpringApplication.run(SpringBoot11Application.class, args);
	}

}
```

- spring-boot-devtools dependency - no need to restart the server each time when we change in appl, the server will automatically restart when we change in appl

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
```

Subject: RE: Reg: Day 19 Notes

Classification: **Confidential**

Handson

1. Create a appl with Employee entity class with id, name, age, gender, department and salary and insert some dummy values in to the table
2. Use JPAQL, Get the details of youngest male employee in the product development department?
3. Using JPAQL, What is the average age of male employees?

JPA Relationship
- used to establish the relationship between entities

1. One to One
- one user has only one vehicle
- We are going to create object of Vehicle class in user class

user Vehicle
userid(pk) name vehicle_id(fk) vehicle_id(pk) vehicle_name

1. Configure db info in persistence.xml

2. Create Vehicle entity class

```
@Entity
@Table(name="veh100")
public class Vehicle {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	@Column(name="vehicle_id")
  private Integer id;
	@Column(name="vehicle_name")
  private String name;
	public Vehicle(Integer id, String name) {
		super();
		this.id = id;
		this.name = name;
	}
	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Vehicle() {
		super();
		// TODO Auto-generated constructor stub
	}


}
```

3. Create User entity class

```
@Entity
@Table(name="user100")
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="user_id")
  private Integer id;
```

```
    @Column(name="user_name")
    private String name;

    @OneToOne
    @JoinColumn(name="vehicle_id")
    private Vehicle vehicle;   //one user has one vehicle

public Integer getId() {
        return id;
}

public void setId(Integer id) {
        this.id = id;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public Vehicle getVehicle() {
        return vehicle;
}

public void setVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
}

public User(Integer id, String name, Vehicle vehicle) {
        super();
        this.id = id;
        this.name = name;
        this.vehicle = vehicle;
}

public User() {
        super();
        // TODO Auto-generated constructor stub
}


}
```

4. Configure entity class in persistence.xml

```
<class>com.pack.User</class>
      <class>com.pack.Vehicle</class>
```

Cascading operation
- used in mapping concept, if we perform operation on one entity class it should be automatically affect the another entity class - 4 types

1. CascadeType.ALL
2. CascadeType.PERSIST
3. CascadeType.UPDATE
4. CascadeType.REMOVE

5. Create main class

```
public class Main1 {

        public static void main(String[] args) {
                EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
                EntityManager em=emf.createEntityManager();
                EntityTransaction et=em.getTransaction();
                et.begin();

                Vehicle v1=new Vehicle();
                v1.setName("Honda");
                Vehicle v2=new Vehicle();
```

```
                v2.setName("Audi");

                User u1=new User();
                u1.setName("Ram");
                u1.setVehicle(v1);
                User u2=new User();
                u2.setName("Raj");
                u2.setVehicle(v2);

                //em.persist(v1);
                //em.persist(v2);
                em.persist(u1);
                em.persist(u2);
                et.commit();
                System.out.println("Success");
                em.close();
                emf.close();
        }

}
```

```
mysql> select * from veh100;
+------------+--------------+
| vehicle_id | vehicle_name |
+------------+--------------+
|         10 | Honda        |
|         12 | Audi         |
+------------+--------------+
2 rows in set (0.01 sec)

mysql> select * from user100;
+---------+-----------+------------+
| user_id | user_name | vehicle_id |
+---------+-----------+------------+
|       9 | Ram       |         10 |
|      11 | Raj       |         12 |
+---------+-----------+------------+
2 rows in set (0.00 sec)
```

2. One to many
   - one user has many vehicle, in this case it create a separate join table

```
user                              Vehicle
userid(pk)   name                vehicle_id(pk)    vehicle_name

user_vehicle
user_id    vehicle_id
```

1. Create Vehicle1 entity class

```
@Entity
@Table(name="veh101")
public class Vehicle1 {
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name="vehicle_id")
   private Integer id;
        @Column(name="vehicle_name")
   private String name;
        public Vehicle1(Integer id, String name) {
                super();
                this.id = id;
                this.name = name;
        }
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
```

```
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Vehicle1() {
		super();
		// TODO Auto-generated constructor stub
	}


}
```

2. Create User1 entity class

```
@Entity
@Table(name="user101")
public class User1 {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="user_id")
  private Integer id;
  @Column(name="user_name")
  private String name;

  //@OneToMany(targetEntity = Vehicle1.class, cascade = CascadeType.ALL)
  @OneToMany(cascade=CascadeType.ALL)
  @JoinTable(name="user_vehicle", joinColumns = @JoinColumn(name="user_id"), inverseJoinColumns = @JoinColumn(name="vehicle_id"))
  private List<Vehicle1> vehicle1=new ArrayList<>();

public Integer getId() {
	return id;
}

public void setId(Integer id) {
	this.id = id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public List<Vehicle1> getVehicle1() {
	return vehicle1;
}

public void setVehicle1(List<Vehicle1> vehicle1) {
	this.vehicle1 = vehicle1;
}

public User1(Integer id, String name, List<Vehicle1> vehicle1) {
	super();
	this.id = id;
	this.name = name;
	this.vehicle1 = vehicle1;
}

public User1() {
	super();
	// TODO Auto-generated constructor stub
}
}
```

3. Configure entity class in persistence.xml

```
<class>com.pack.User1</class>
        <class>com.pack.Vehicle1</class>
```

4. Create main class

```
public class Main1 {

        public static void main(String[] args) {
                EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
                EntityManager em=emf.createEntityManager();
                EntityTransaction et=em.getTransaction();
                et.begin();

                Vehicle1 v1=new Vehicle1();
                v1.setName("BMW");
                Vehicle1 v2=new Vehicle1();
                v2.setName("Benz");

          List<Vehicle1> list=new ArrayList<>();
          list.add(v1);
          list.add(v2);

                User1 u1=new User1();
                u1.setName("Ramu");
                u1.setVehicle1(list);

                em.persist(u1);

                et.commit();
                System.out.println("Success");
                em.close();
                emf.close();
        }

}
```

```
mysql> select * from user101;
+---------+-----------+
| user_id | user_name |
+---------+-----------+
|     13 | Ramu      |
+---------+-----------+
1 row in set (0.00 sec)

mysql> select * from veh101;
+------------+--------------+
| vehicle_id | vehicle_name |
+------------+--------------+
|        14 | BMW          |
|        15 | Benz         |
+------------+--------------+
2 rows in set (0.01 sec)

mysql> select * from user101_veh101;
+---------------+---------------------+
| User1_user_id | vehicle1_vehicle_id |
+---------------+---------------------+
|          13 |                14 |
|          13 |                15 |
+---------------+---------------------+
2 rows in set (0.01 sec)
```

3. Many to one
   - many user will have one vehicle

4. Many to many
   - many user will have many vehicle

1. Create Vehicle2 entity class

```
@Entity
@Table(name="veh102")
public class Vehicle2 {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	@Column(name="vehicle_id")
  private Integer id;
	@Column(name="vehicle_name")
  private String name;

	@ManyToMany(targetEntity=User2.class)
	private List<User2> user=new ArrayList<>();

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public List<User2> getUser() {
		return user;
	}

	public void setUser(List<User2> user) {
		this.user = user;
	}

	public Vehicle2(Integer id, String name, List<User2> user) {
		super();
		this.id = id;
		this.name = name;
		this.user = user;
	}

	public Vehicle2() {
		super();
		// TODO Auto-generated constructor stub
	}


}
```

2. Craete User2 entity class

```
@Entity
@Table(name="user102")
public class User2 {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="user_id")
  private Integer id;
  @Column(name="user_name")
  private String name;

  @ManyToMany(targetEntity = Vehicle2.class, cascade=CascadeType.ALL)
  private List<Vehicle2> vehicle=new ArrayList<>();

public Integer getId() {
	return id;
```

```
}

public void setId(Integer id) {
	this.id = id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public List<Vehicle2> getVehicle() {
	return vehicle;
}

public void setVehicle(List<Vehicle2> vehicle) {
	this.vehicle = vehicle;
}

public User2(Integer id, String name, List<Vehicle2> vehicle) {
	super();
	this.id = id;
	this.name = name;
	this.vehicle = vehicle;
}

public User2() {
	super();
	// TODO Auto-generated constructor stub
}
 }
```

3. Configure entity class in xml file

```
<class>com.pack.User2</class>
	<class>com.pack.Vehicle2</class>
```

4. Create main class

```
public class Main1 {

	public static void main(String[] args) {
		EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
		EntityManager em=emf.createEntityManager();
		EntityTransaction et=em.getTransaction();
		et.begin();

		Vehicle2 v1=new Vehicle2();
		v1.setName("Duster");
		Vehicle2 v2=new Vehicle2();
		v2.setName("Ford");

	  List<Vehicle2> list1=new ArrayList<>();
	  list1.add(v1);
	  list1.add(v2);

	  List<Vehicle2> list2=new ArrayList<>();
	  list2.add(v1);
	  list2.add(v2);

		User2 u1=new User2();
		u1.setName("Amy");
		u1.setVehicle(list1);

		User2 u2=new User2();
		u2.setName("Jack");
		u2.setVehicle(list2);

		em.persist(u1);
```

```
                em.persist(u2);

                et.commit();
                System.out.println("Success");
                em.close();
                emf.close();
        }

}
```

JPA Inheritance
- entity class can also be inherited - 3 types

1. Single Table/Table per class inheritance
   - For all entity class it will create a common single table

@DiscriminatorColumn - used to differentiate the type of employee
@DiscriminatorValue - used to provide value for that column

```
@Entity
@Table(name="emp100")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="empType", discriminatorType = DiscriminatorType.STRING)
@DiscriminatorValue(value="employee")
public class Employee {
  @Id
  private Integer id;
  private String name;
public Integer getId() {
        return id;
}
public void setId(Integer id) {
        this.id = id;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public Employee(Integer id, String name) {
        super();
        this.id = id;
        this.name = name;
}
public Employee() {
        super();
        // TODO Auto-generated constructor stub
}


}

@Entity
@DiscriminatorValue(value="reg_employee")
public class RegularEmployee extends Employee{
  private Double salary;
  private Double bonus;
public Double getSalary() {
        return salary;
}
public void setSalary(Double salary) {
        this.salary = salary;
}
public Double getBonus() {
        return bonus;
}
public void setBonus(Double bonus) {
        this.bonus = bonus;
}
public RegularEmployee(Double salary, Double bonus) {
```

```
		super();
		this.salary = salary;
		this.bonus = bonus;
	}
	public RegularEmployee() {
		super();
		// TODO Auto-generated constructor stub
	}

	}

	@Entity
	@DiscriminatorValue(value="cont_employee")
	public class ContractEmployee extends Employee{
	   private Double payPerHour;
	   private Integer duration;
		public Double getPayPerHour() {
			return payPerHour;
		}
		public void setPayPerHour(Double payPerHour) {
			this.payPerHour = payPerHour;
		}
		public Integer getDuration() {
			return duration;
		}
		public void setDuration(Integer duration) {
			this.duration = duration;
		}
		public ContractEmployee(Double payPerHour, Integer duration) {
			super();
			this.payPerHour = payPerHour;
			this.duration = duration;
		}
		public ContractEmployee() {
			super();
			// TODO Auto-generated constructor stub
		}

	}
```

2. Create main class

```
	public class Main1 {

		public static void main(String[] args) {
			EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
			EntityManager em=emf.createEntityManager();
			EntityTransaction et=em.getTransaction();
			et.begin();

			Employee e1=new Employee();
			e1.setId(100);
			e1.setName("Ram");

			RegularEmployee r1=new RegularEmployee();
			r1.setId(101);
			r1.setName("Sam");
			r1.setSalary(20000.0);
			r1.setBonus(2000.0);

			ContractEmployee c1=new ContractEmployee();
			c1.setId(102);
			c1.setName("Jam");
			c1.setPayPerHour(200.0);
			c1.setDuration(2);

			em.persist(e1);
			em.persist(r1);
			em.persist(c1);
			et.commit();
			System.out.println("Success");
```

```
                em.close();
                emf.close();
        }

}
```

```
mysql> select * from emp100;
+---------------+-----+------+-------+--------+----------+------------+
| empType       | id  | name | bonus | salary | duration | payPerHour |
+---------------+-----+------+-------+--------+----------+------------+
| employee      | 100 | Ram  |  NULL |   NULL |     NULL |       NULL |
| reg_employee  | 101 | Sam  |  2000 |  20000 |     NULL |       NULL |
| cont_employee | 102 | Jam  |  NULL |   NULL |        2 |        200 |
+---------------+-----+------+-------+--------+----------+------------+
3 rows in set (0.00 sec)
```

2. Table per concrete class inheritance
   - For each entity class it creates separate table

```
@Entity
@Table(name="emp101")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Employee {
  @Id
  private Integer id;
  private String name;
public Integer getId() {
        return id;
}
public void setId(Integer id) {
        this.id = id;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public Employee(Integer id, String name) {
        super();
        this.id = id;
        this.name = name;
}
public Employee() {
        super();
        // TODO Auto-generated constructor stub
}


}


@Entity
@Table(name="regemp101")
@AttributeOverrides({
        @AttributeOverride(name="id",column=@Column(name="id")),
    @AttributeOverride(name="name",column=@Column(name="name")),
})
public class RegularEmployee extends Employee{
  private Double salary;
  private Double bonus;
public Double getSalary() {
        return salary;
}
public void setSalary(Double salary) {
        this.salary = salary;
}
public Double getBonus() {
        return bonus;
}
```

```
	public void setBonus(Double bonus) {
		this.bonus = bonus;
	}
	public RegularEmployee(Double salary, Double bonus) {
		super();
		this.salary = salary;
		this.bonus = bonus;
	}
	public RegularEmployee() {
		super();
		// TODO Auto-generated constructor stub
	}

	}

	@Entity
	@Table(name="contemp101")
	@AttributeOverrides({
		@AttributeOverride(name="id",column=@Column(name="id")),
	    @AttributeOverride(name="name",column=@Column(name="name")),
	})
	public class ContractEmployee extends Employee{
	   private Double payPerHour;
	   private Integer duration;
		public Double getPayPerHour() {
			return payPerHour;
		}
		public void setPayPerHour(Double payPerHour) {
			this.payPerHour = payPerHour;
		}
		public Integer getDuration() {
			return duration;
		}
		public void setDuration(Integer duration) {
			this.duration = duration;
		}
		public ContractEmployee(Double payPerHour, Integer duration) {
			super();
			this.payPerHour = payPerHour;
			this.duration = duration;
		}
		public ContractEmployee() {
			super();
			// TODO Auto-generated constructor stub
		}

	}

	public class Main1 {

		public static void main(String[] args) {
			EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
			EntityManager em=emf.createEntityManager();
			EntityTransaction et=em.getTransaction();
			et.begin();

			Employee e1=new Employee();
			e1.setId(100);
			e1.setName("Ram");

			RegularEmployee r1=new RegularEmployee();
			r1.setId(101);
			r1.setName("Sam");
			r1.setSalary(20000.0);
			r1.setBonus(2000.0);

			ContractEmployee c1=new ContractEmployee();
			c1.setId(102);
			c1.setName("Jam");
			c1.setPayPerHour(200.0);
			c1.setDuration(2);
```

```
                em.persist(e1);
                em.persist(r1);
                em.persist(c1);
                et.commit();
                System.out.println("Success");
                em.close();
                emf.close();
        }

}
```

```
mysql> select * from emp101;
+-----+------+
| id  | name |
+-----+------+
| 100 | Ram  |
+-----+------+
1 row in set (0.00 sec)

mysql> select * from regemp101;
+-----+------+-------+--------+
| id  | name | bonus | salary |
+-----+------+-------+--------+
| 101 | Sam  |  2000 |  20000 |
+-----+------+-------+--------+
1 row in set (0.01 sec)

mysql> select * from contemp101;
+-----+------+----------+------------+
| id  | name | duration | payPerHour |
+-----+------+----------+------------+
| 102 | Jam  |        2 |        200 |
+-----+------+----------+------------+
1 row in set (0.01 sec)
```

3. Joined/Table per subclass inheritance
   - used to join the table based on primary key and foreign key relationship

```
@Entity
@Table(name="emp102")
@Inheritance(strategy=InheritanceType.JOINED)
public class Employee {
  @Id
  private Integer id;
  private String name;
public Integer getId() {
        return id;
}
public void setId(Integer id) {
        this.id = id;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public Employee(Integer id, String name) {
        super();
        this.id = id;
        this.name = name;
}
public Employee() {
        super();
        // TODO Auto-generated constructor stub
}


}
```

```
@Entity
@Table(name="regemp102")
@PrimaryKeyJoinColumn(name="id")
public class RegularEmployee extends Employee{
  private Double salary;
  private Double bonus;
public Double getSalary() {
	return salary;
}
public void setSalary(Double salary) {
	this.salary = salary;
}
public Double getBonus() {
	return bonus;
}
public void setBonus(Double bonus) {
	this.bonus = bonus;
}
public RegularEmployee(Double salary, Double bonus) {
	super();
	this.salary = salary;
	this.bonus = bonus;
}
public RegularEmployee() {
	super();
	// TODO Auto-generated constructor stub
}

}


@Entity
@Table(name="contemp102")
@PrimaryKeyJoinColumn(name="id")
public class ContractEmployee extends Employee{
   private Double payPerHour;
   private Integer duration;
	public Double getPayPerHour() {
		return payPerHour;
	}
	public void setPayPerHour(Double payPerHour) {
		this.payPerHour = payPerHour;
	}
	public Integer getDuration() {
		return duration;
	}
	public void setDuration(Integer duration) {
		this.duration = duration;
	}
	public ContractEmployee(Double payPerHour, Integer duration) {
		super();
		this.payPerHour = payPerHour;
		this.duration = duration;
	}
	public ContractEmployee() {
		super();
		// TODO Auto-generated constructor stub
	}

}


mysql> select * from emp102;
+-----+------+
| id  | name |
+-----+------+
| 100 | Ram  |
| 101 | Sam  |
| 102 | Jam  |
```

```
+-----+------+
3 rows in set (0.00 sec)

mysql> select * from regemp102;
+-------+--------+-----+
| bonus | salary | id  |
+-------+--------+-----+
|  2000 |  20000 | 101 |
+-------+--------+-----+
1 row in set (0.01 sec)

mysql> select * from contemp102;
+----------+------------+-----+
| duration | payPerHour | id  |
+----------+------------+-----+
|        2 |        200 | 102 |
+----------+------------+-----+
1 row in set (0.01 sec)
```

JPA QL - JPA Query Language

- Database independent query because we are going to query the entity class using Query interface

```
Query q=em.createQuery(String query);
```

1. Create Person entity class

```
@Entity
@Table(name="person100")
public class Person {
  @Id
  private Integer id;
  private String name;
  private String address;
  private Integer age;
  private String city;
public Person(Integer id, String name, String address, Integer age, String city) {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        this.age = age;
        this.city = city;
}
public Integer getId() {
        return id;
}
public void setId(Integer id) {
        this.id = id;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public String getAddress() {
        return address;
}
public void setAddress(String address) {
        this.address = address;
}
public Integer getAge() {
        return age;
}
public void setAge(Integer age) {
        this.age = age;
}
public String getCity() {
        return city;
}
```

```
public void setCity(String city) {
	this.city = city;
}
public Person() {
	super();
	// TODO Auto-generated constructor stub
}


}
```

2. Configure entity class in xml file

3. Create main class

```
public class Main2 {

	public static void main(String[] args) {
		EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
		EntityManager em=emf.createEntityManager();
		EntityTransaction et=em.getTransaction();
		et.begin();

		/*Person p1=new Person(100,"Ram","ABC Street",23,"Chennai");
		em.persist(p1);
		Person p2=new Person(101,"Sam","XYZ Street",24,"Mumbai");
		em.persist(p2);
		Person p3=new Person(102,"Raj","PQR Street",21,"Pune");
		em.persist(p3);
		Person p4=new Person(103,"Tam","RTY Street",22,"Delhi");
		em.persist(p4);
		Person p5=new Person(104,"Jam","BGR Street",33,"Noida");
		em.persist(p5);
		Person p6=new Person(105,"Jim","ABd Street",21,"Bangalore");
		em.persist(p6);*/

		//1. Select all properties from object
		/*Query q=em.createQuery("select p from Person p");
		List<Person> l=q.getResultList();
		//l.forEach(System.out::println);
		for(Person p:l)
			System.out.println(p.getName()+" "+p.getAddress());*/

		//2. Select only particular properties from object
		/*Query q=em.createQuery("select p.name,p.age,p.city from Person p");
		List l=q.getResultList();
		Iterator i=l.iterator();
		while(i.hasNext()) {
			Object[] o=(Object[])i.next();
			System.out.println(o[0]+" "+o[1]+" "+o[2]);
		}*/

		//3. Select only single property
		/*Query q=em.createQuery("select p.age from Person p");
		List<Integer> l=q.getResultList();
		l.forEach(System.out::println);*/

		//4. Select all property from object but single value
		/*Query q=em.createQuery("select p from Person p where p.id=101");
		Person p=(Person)q.getSingleResult();
		System.out.println(p.getAddress()+" "+p.getCity());*/

		/*Scanner sc=new Scanner(System.in);
	System.out.println("Enter id");
	int pid=sc.nextInt();*/

	//5. Named parameter - using : with any name - :a, :xyz,:cb
	/*Query q=em.createQuery("select p from Person p where p.id=:abc and p.city=:xyz");
	q.setParameter("abc", pid);
	q.setParameter("xyz","Delhi");
		Person p=(Person)q.getSingleResult();
```

```
                System.out.println(p.getAddress()+" "+p.getCity());*/

        //6. Positional Parameter - ?1,?2 etc
        /*Query q=em.createQuery("select p from Person p where p.id=?1 and p.city=?2");
        q.setParameter(1,102);
        q.setParameter(2,"Pune");
                Person p=(Person)q.getSingleResult();
                System.out.println(p.getAddress()+" "+p.getCity());*/

                //7. Executing DML queries using executeUpdate()
                /*Query q=em.createQuery("update Person p set p.address=:addr, p.city=:city where p.id=:id");
        q.setParameter("addr","Pink Street");
        q.setParameter("city","Jaipur");
        q.setParameter("id",102);
        int i=q.executeUpdate();
        System.out.println(i+" rows updated");*/

                //8. Pagination - limiting the records
                Query q=em.createQuery("select p from Person p");
        q.setFirstResult(2);  //starts from 0th position, it will skip 2 records
                q.setMaxResults(2);  //print only 2 records
                List<Person> l=q.getResultList();
                for(Person p:l)
                        System.out.println(p.getName());


                et.commit();
                System.out.println("Success");
                em.close();
                emf.close();

        }

}
```

Subject: RE: Reg: Day 18 Notes

Classification: **Confidential**

Handson

1. Write hibernate appl to insert Person details like id,name,salary, address into database

ORM Framework
- Object Relational Mapping - used to persist the data into database

JDBC - Java Database Connectivity
EJB 2.x Entity bean - Failed in performance
EJB 3.X - JPA - Java Persistence API

JPA is a specification to persist the data into database and it uses some ORM concept
It requires some implementation using ORM tools like Hibernate, iBatis, Toplink, JDO etc

Object Relational Mapping(ORM) used to map POJO class(entity/persistent class) to the columns of database table

1. open source and lightweight
2. Database independent query
   JPAQL(Java Persistent Query Language) - query the entity class
   SQL - query the table directly
3. Automatic table creation
4. Fast performance - uses cache framework
5. Simplifies complex join
6. Hibernate will automatically generate the queries

1. Configure database info and hibernate properties

```
<!--Database info--!>
<property name="connection.url">jdbc:mysql://localhost:3306/jpa</property>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

<property name="connection.username">root</property>
<property name="connection.password">root</property>

<!--Hibernate properties --!>
<property name="hbm2ddl.auto">create/update/create-drop/validate</property> - will automatically create the table - 4 values
1. create - each time it will create new table
2. update - it will create table for first time, and next time it will update the table(ie) append the data
3. create-drop - it drop the entire db and create a new table
4. validate - it dosent do anything but it just validate the existing table

<property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
- we can communicate with db using queries or using predefined methods
- we wont write any sql queries, instead we use some predefined methods to processing in the database, so hibernate internally will generate sql queries related to that particular method based on particular database
<property name="show_sql">true</property>
- used to display generated sql query in console in single line
<property name="format_sql">true</property>
- used to format and display generated sql query in console
<property name="use_sql_comments">true</property>
- used to define comments for the generated sql query in console

JPA - Java Persistence API
- It is a specification in Java used to persist data between java object and relational db
- collection of classes and interface to store data into database
- present in javax.persistence.*

Entity/persistent class(user defined class) - persistent object that is stored inside db - POJO class

1. EntityManagerFactory interface - used to create EntityManager
- EntityManager createEntityManager()
2. EntityManager interface - core interface used to persist the data into db - acts as interface between appl and database
- void persist(Object o) - insert the data into db if primary key does not exists, if primary key exists it perform update operation - generate insert query
- Object find(Serializable s, Object key) - select the data from db - generate select query
- void remove(Object o) - delete data from db - generate delete query
- void close()
- boolean contains(Object o)
- Query createQuery(String query)
- Query createNamedQuery(String query)
- Query createNativeQuery(String query)
- CriteriaBuilder getCriteriaBuilder()
- EntityTransaction getTransaction() - create a transaction
3. EntityTransaction interface - used to maintain transaction - commit(), rollback()
4. Query interface - used to write database independent query

1. Create maven project with 2 dependency(hibernate, mysql db)

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.4.24.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.19</version>
</dependency>

2. Configure db info in persistence.xml inside resources/META-INF

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence version="2.0">
<persistence-unit name="student_pu" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>com.pack.Student</class>
<properties>
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/prejoiners"/>
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

```
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>

            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.use_sql_comments" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

3. Create Entity class using JPA - simple pojo - use jpa concept

```
import javax.persistence.*;

@Entity - indicate it is entity class which will store in db
@Table(name="stud100") - Optional - used to map the entity class with table, if not given then it create table in name of entity class
public class Student {
    @Id - indicates it is a primary key
    @GeneratedValue(strategy=GenerationType.AUTO/IDENTITY/SEQUENCE/TABLE) - To indicate primary key value to be autogenerated, if we not
provided @GeneratedValue we have to provide value for primary key

@Column(name="studid",length=20,nullable="true/false",unique="true/false",insertable="true/false",updatable="true/false",scale="7",precision="2")
    private Integer id;

    @Column(name="stuname")
    private String name;

    private Integer age;

    @Temporal(TemporalType.DATE/TIME/TIMESTAMP) - used to map java date field to sql date field
    private Date dob;

    //private String gender;
    @Enumerated(EnumType.ORDINAL(0,1)/STRING(MALE,FEMALE)) - used to map enum field to table
    private Gender gender;  //here gender is enum

    @Lob - used to map large objects to table
    @Column(name="photo",ColumnDefination="BLOB")
    private byte[] myImage;

    @Transient - used to ignore this field at time of persisting
    private boolean status;

    //getters and setters
}

@Column(name="sal",scale="7",precision="2")
private Double salary; //20000.50


enum Gender { MALE, FEMALE}

@Entity
@Table(name="stud100")
public class Student {
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name="stuid")
    private Integer id;
        @Column(name="stuname")
    private String name;
    private Integer age;
    @Temporal(TemporalType.DATE)
    private Date dob;
    @Enumerated(EnumType.ORDINAL)
    private Gender gender;
    @Transient
```

```
    private boolean status;

    //getters, setters,constructors

}
```

4. Configure entity class in persistence.xml

```
<class>com.pack.Student</class>
```

5. Create main class

1. Create EntityManagerFactory - get all db info from persistence.xml
2. Create EntityManager
3. Create EntityTransaction

@DynamicInsert - if we want to generate sql queries only for the column u r inserting
@DynamicUpdate - if we want to generate sql queries only for the column u r updating

```
public class Main {
	public static void main(String[] args) {

		EntityManagerFactory emf=Persistence.createEntityManagerFactory("student_pu");
		EntityManager em=emf.createEntityManager();
		EntityTransaction et=em.getTransaction();
		et.begin();
		/*Student st1=new Student("Tim",23,LocalDate.parse("2000-10-23"),Gender.MALE);
		em.persist(st1);*/

		/*Student st=(Student)em.find(Student.class, 5);
		st.setGender(Gender.FEMALE);
		em.persist(st);*/

		/*Student st=(Student)em.find(Student.class, 5);
		em.remove(st);*/

		/*Student st=new Student();
		st.setName("Amy");
		st.setAge(23);
		st.setDob(LocalDate.parse("1990-02-23"));
		st.setGender(Gender.FEMALE);
		em.persist(st);*/

		/*Student st=new Student();
		st.setName("July");
		st.setDob(LocalDate.parse("1993-02-23"));
		st.setGender(Gender.FEMALE);
		em.persist(st);*/

		Student st=(Student)em.find(Student.class, 8);
		st.setAge(28);
		em.persist(st);
		et.commit();
		System.out.println("Success");
		em.close();
		emf.close();
	}
}
```

Subject: RE: Reg: Day 18 Notes

Classification: **Confidential**

Spring Annotations
1. @Configuration
2. @Bean

```
public class Example3 {
  private String message;
```

```
public Example3(String message) {
	super();
	this.message = message;
}

public Example3() {
	super();
	// TODO Auto-generated constructor stub
}

@Override
public String toString() {
	return "Example3 [message=" + message + "]";
}


}


@Configuration
public class ExampleConfiguration {

	@Bean
	@Scope("prototype")
	public Example1 example1() {
		return new Example1();
	}

	@Bean
	public Example2 example2() {
		return new Example2();
	}

	@Bean
	public Example3 example3() {
		return new Example3("Spring");
	}
	@Bean
	@Primary
	public Example3 example31() {
		return new Example3("Springboot");
	}
	@Bean
	public Example3 example32() {
		return new Example3("Struts");
	}
}

@Autowired
   - perform autowiring based on by type
   - used above only setter method or constructor or properties
   - If we use autowire by annotation, in order to make ur spring to understand the annotation we have to configure <context:annotation-config/> in xml
file

public class Address {
    private String city;
    private String state;
    private String pincode;
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public String getState() {
		return state;
	}
	public void setState(String state) {
		this.state = state;
	}
```

```
		public String getPincode() {
			return pincode;
		}
		public void setPincode(String pincode) {
			this.pincode = pincode;
		}


}

public class Student {
    private Integer id;
    private String name;
    @Autowired
    private Address address;  //referred another bean
		public Integer getId() {
			return id;
		}
		public void setId(Integer id) {
			this.id = id;
		}
		public String getName() {
			return name;
		}
		public void setName(String name) {
			this.name = name;
		}
		public Address getAddress() {
			return address;
		}
		//@Autowired
		public void setAddress(Address address) {
			this.address = address;
		}

}

<context:annotation-config/>
   <bean id="addr" class="com.pack.Address">
      <property name="city" value="Chennai"/>
      <property name="state" value="Tamilnadu"/>
      <property name="pincode" value="600011"/>
   </bean>
   <bean id="student" class="com.pack.Student">
       <property name="id" value="1000"/>
       <property name="name" value="John"/>
      <!--  <property name="address" ref="addr"/> -->
   </bean>

ApplicationContext context=new ClassPathXmlApplicationContext("bean.xml");
			Student st=(Student)context.getBean("student");
			System.out.println(st.getName()+" "+st.getAddress().getCity());

@Qualifier
   - If same bean is configured multiple times in xml file which bean has to be injected while using @Autowired is decided using @Qualifier

<context:annotation-config/>
   <bean id="addr" class="com.pack.Address">
      <property name="city" value="Chennai"/>
      <property name="state" value="Tamilnadu"/>
      <property name="pincode" value="600011"/>
   </bean>
   <bean id="addr1" class="com.pack.Address">
      <property name="city" value="Pune"/>
      <property name="state" value="Maharastra"/>
      <property name="pincode" value="200011"/>
   </bean>
   <bean id="student" class="com.pack.Student">
       <property name="id" value="1000"/>
       <property name="name" value="John"/>
      <!--  <property name="address" ref="addr"/> -->
```

```
</bean>

public class Student {
    private Integer id;
    private String name;
    @Autowired
    @Qualifier("addr1")
    private Address address;  //referred another bean
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Address getAddress() {
                return address;
        }
        //@Autowired
        public void setAddress(Address address) {
                this.address = address;
        }

}
```

@Required
- It indicates that the property should be complusorly injected in xml file
- always used only above setter method

```
public class Student {
    private Integer id;
    private String name;
    @Autowired
    @Qualifier("addr1")
    private Address address;  //referred another bean
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        @Required
        public void setName(String name) {
                this.name = name;
        }
        public Address getAddress() {
                return address;
        }
        //@Autowired
        public void setAddress(Address address) {
                this.address = address;
        }

}
```

BeanFactory is Lazy loading - it creates the bean only when we call getBean()

ApplicationContext is eager loading - preloads all the beans at time of startup itself, so we have to convert ApplicationContext to lazy loading
1. using lazy-init="true" in xml file
2. Using @Lazy annotation

```
public class Sample1 {
```

```
	public Sample1() {
		System.out.println("Sample1 bean is created");
	}

}

public class Sample2 {
  private Sample1 sample1;

	public Sample1 getSample1() {
		return sample1;
	}

	public void setSample1(Sample1 sample1) {
		this.sample1 = sample1;
	}

	public Sample2() {
		System.out.println("Sample2 bean is created");
	}

}

<bean id="sample1" class="com.pack.Sample1" lazy-init="true"/>
  <bean id="sample2" class="com.pack.Sample2" lazy-init="true">
    <property name="sample1" ref="sample1"/>
  </bean>

public class Main3 {

	public static void main(String[] args) {
		//Resource res=new ClassPathResource("bean.xml");
		 //BeanFactory factory=new XmlBeanFactory(res);
		 //Sample2 s=(Sample2)factory.getBean("sample2");

		 ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");
		 Sample2 s=(Sample2)ctx.getBean("sample2");
	}

}
```

Lifecycle of spring bean
1. Using InitializingBean and DisposableBean interface

InitializingBean interface- public void afterPropertiesSet() - invoked whenever a bean is instantiated when getBean() is called
DisposableBean interface - public void destroy() - invoked whenever ur bean is destroyed

ConfigurableApplicationContext interface
- abstract class AbstractApplicationContext implements ConfigurableApplicationContext interface
void registerShutDownHook() - destroy the bean
  or
void close() - destroy the bean

```
public class Person implements InitializingBean,DisposableBean {
  private String name;

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public void destroy() throws Exception {
	System.out.println("Destroying person bean");
}

public void afterPropertiesSet() throws Exception {
  System.out.println("Initializing person bean "+name);
```

```
}

}
```

```
<bean id="person" class="com.pack.Person">
      <property name="name" value="Ram"/>
   </bean>

public class Main4 {

        public static void main(String[] args) {
                ConfigurableApplicationContext ctx=new ClassPathXmlApplicationContext("lifecycle.xml");
                Person p=(Person)ctx.getBean("person");
                System.out.println(p.getName());
                //ctx.registerShutdownHook();
                ctx.close();
        }

}
```

2. custom init and destroy method
   - using init-method and destroy-method in xml file for particular bean class

```
public class Example {
  private String name;

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

  public void customInit() {
          System.out.println("Initializing example bean");
  }

  public void customDestroy() {
          System.out.println("Destroying example bean");
  }
}

<bean id="example" class="com.pack.Example" init-method="customInit" destroy-method="customDestroy">
     <property name="name" value="Sam"/>
   </bean>

ConfigurableApplicationContext ctx=new ClassPathXmlApplicationContext("lifecycle.xml");
                Example e=(Example)ctx.getBean("example");
                System.out.println(e.getName());
                ctx.registerShutdownHook();
```

3. default-init-method="customInit" and default-destroy-method="customDestroy" in beans tag, so that it is applicable for all beans in xml file

4. Using @PostConstruct and @PreDestroy annotation

```
<dependency>
                    <groupId>javax.annotation</groupId>
                    <artifactId>javax.annotation-api</artifactId>
                    <version>1.3.2</version>
                </dependency>
```

Spring sterotype annotations
1. @Component - it allows spring to identify that class as custom bean so that it can be injected in other prg using @Autowired
2. @Service - it is a bean class that contains business logic, so that it can be injected in other prg using @Autowired
3. @Repository - it is a bean class that interacts with database
4. @Controller - used to define a controller class that accept the request and gives the response

Subject: RE: Reg: Day 17 Notes

Classification: **Confidential**

Scope
- life time of bean object - 5 types

1. singleton - default - how many bean object we are creating it will have same object
2. prototype - how many bean object we are creating it will have different reference
3. session
4. request
5. global-session

```
 <bean id="emp" class="com.pack.Employee" scope="prototype">
     <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>
     <constructor-arg  index="1" type="java.lang.String" value="Sam"/>
     <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>
     <constructor-arg  index="3" type="int" value="20"/>
  </bean>

ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
                    Employee e=(Employee)context.getBean("emp");
                    System.out.println(e);

                    Employee e1=(Employee)context.getBean("emp");
                    System.out.println(e1);

                    Employee e2=(Employee)context.getBean("emp");
                    System.out.println(e2);
```

Injection Collections
- apart from primitive datatype we can also inject collections like List,Set, Map,Properties

```
public class SampleCollection {
    private List addrList;
    private Set addrSet;
    private Map addrMap;
    private Properties addrProp;
        public List getAddrList() {
                return addrList;
        }
        public void setAddrList(List addrList) {
                this.addrList = addrList;
        }
        public Set getAddrSet() {
                return addrSet;
        }
        public void setAddrSet(Set addrSet) {
                this.addrSet = addrSet;
        }
        public Map getAddrMap() {
                return addrMap;
        }
        public void setAddrMap(Map addrMap) {
                this.addrMap = addrMap;
        }
        public Properties getAddrProp() {
                return addrProp;
        }
        public void setAddrProp(Properties addrProp) {
                this.addrProp = addrProp;
        }

}


<bean id="collection" class="com.pack.SampleCollection">
       <property name="addrList">
          <list>
            <value>Ram</value>
            <value>Sam</value>
```

```
            <value>Raj</value>
            <value>Ram</value>
          </list>
        </property>
        <property name="addrSet">
          <set>
            <value>Ram</value>
            <value>Sam</value>
            <value>Raj</value>
            <value>Ram</value>
          </set>
        </property>
        <property name="addrMap">
          <map>
            <entry key="1" value="Ram"/>
            <entry key="2" value="Sam"/>
            <entry key="3" value="Tam"/>
            <entry key="4" value="Raj"/>
          </map>
        </property>
        <property name="addrProp">
          <props>
            <prop key="one">Ram</prop>
            <prop key="two">Sam</prop>
            <prop key="three">Raj</prop>
          </props>
        </property>
    </bean>

public class Main1 {
        public static void main(String[] args) {
                ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");
                SampleCollection s=(SampleCollection)ctx.getBean("collection");
                System.out.println(s.getAddrList());
                System.out.println(s.getAddrSet());
                System.out.println(s.getAddrMap());
                System.out.println(s.getAddrProp());
        }
}

ref attribute
   - used to refer one bean into another bean
   - ref="id of the bean"

public class Address {
   private String city;
   private String state;
   private String pincode;
        public String getCity() {
                return city;
        }
        public void setCity(String city) {
                this.city = city;
        }
        public String getState() {
                return state;
        }
        public void setState(String state) {
                this.state = state;
        }
        public String getPincode() {
                return pincode;
        }
        public void setPincode(String pincode) {
                this.pincode = pincode;
        }

}

public class Student {
   private Integer id;
```

```
    private String name;
    private Address address;  //referred another bean
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Address getAddress() {
                return address;
        }
        public void setAddress(Address address) {
                this.address = address;
        }

}

<bean id="addr" class="com.pack.Address">
      <property name="city" value="Chennai"/>
      <property name="state" value="Tamilnadu"/>
      <property name="pincode" value="600011"/>
   </bean>
   <bean id="student" class="com.pack.Student">
       <property name="id" value="1000"/>
       <property name="name" value="John"/>
       <property name="address" ref="addr"/>
   </bean>

public class Main1 {
        public static void main(String[] args) {
                ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");
                Student st=(Student)ctx.getBean("student");
                System.out.println(st.getId()+" "+st.getName()+" "+st.getAddress().getCity());
        }
}


wiring - refering one bean into another bean

Autowiring
  - used only in case of referring one bean into another bean
  - Instead of using ref attribute we can go for autowiring to inject one bean into another bean automatically

Types
1. autowire=none - default - where we have to do injection using ref attribute

public class Match {
   private String location;
   private String distance;
public String getLocation() {
        return location;
}
public void setLocation(String location) {
        this.location = location;
}
public String getDistance() {
        return distance;
}
public void setDistance(String distance) {
        this.distance = distance;
}

}

public class Player {
```

```
  private Integer playerId;
  private String name;
  private Match match;
public Integer getPlayerId() {
        return playerId;
}
public void setPlayerId(Integer playerId) {
        this.playerId = playerId;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public Match getMatch() {
        return match;
}
public void setMatch(Match match) {
        this.match = match;
}

}
```

```
<bean id="match" class="com.pack.Match">
    <property name="location" value="Chennai"/>
    <property name="distance" value="1000"/>
  </bean>
  <bean id="player" class="com.pack.Player">
    <property name="playerId" value="100"/>
    <property name="name" value="Dhoni"/>
    <property name="match" ref="match"/>
  </bean>
```

```
public class Main1 {
        public static void main(String[] args) {
                ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");
                Player p=(Player)ctx.getBean("player");
                System.out.println(p.getName()+" plays at "+p.getMatch().getLocation());
        }
}
```

2. autowire=byName - name of the property = id of the bean in xml file, then it will do autowire by name

```
<bean id="match" class="com.pack.Match">
    <property name="location" value="Chennai"/>
    <property name="distance" value="1000"/>
  </bean>
  <bean id="player" class="com.pack.Player" autowire="byName">
    <property name="playerId" value="100"/>
    <property name="name" value="Dhoni"/>
  </bean>
```

3. autowire=byType - if datatype of the property should be somewhere configured in xml file, then it will do autowire by type

```
<bean id="match" class="com.pack.Match">
    <property name="location" value="Chennai"/>
    <property name="distance" value="1000"/>
  </bean>
  <bean id="player" class="com.pack.Player" autowire="byType">
    <property name="playerId" value="100"/>
   <property name="name" value="Dhoni"/>
  </bean>
```

4. autowire=constructor - similar to byType or byname but it will do injection through constructor

```
public class Match {
  private String location;
  private String distance;
public Match(String location, String distance) {
        super();
```

```
		this.location = location;
		this.distance = distance;
	}
	public Match() {
		super();
		// TODO Auto-generated constructor stub
	}
	@Override
	public String toString() {
		return "Match [location=" + location + ", distance=" + distance + "]";
	}


	}

	public class Player {
	  private Integer playerId;
	  private String name;
	  private Match match1;
	public Player(Integer playerId, String name, Match match1) {
		super();
		this.playerId = playerId;
		this.name = name;
		this.match1 = match1;
	}
	public Player() {
		super();
		// TODO Auto-generated constructor stub
	}
	@Override
	public String toString() {
		return "Player [playerId=" + playerId + ", name=" + name + ", match1=" + match1 + "]";
	}

	}

	<bean id="match" class="com.pack.Match">
	    <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>
	    <constructor-arg index="1" type="java.lang.String" value="1000"/>
	  </bean>
	  <bean id="player" class="com.pack.Player" autowire="constructor">
	    <constructor-arg index="0" type="java.lang.Integer" value="1001"/>
	    <constructor-arg index="1" type="java.lang.String" value="Kohli"/>
	  </bean>

	public class Main1 {
		public static void main(String[] args) {
			ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");
			Player p=(Player)ctx.getBean("player");
			System.out.println(p);
		}
	}

	autowire-candidate=false
	  - If same bean is configured multiple times in xml file, which bean class has to be injected, it is defined using autowire-candidate property
	  - So which bean we dont want to use we have to define autowire-candidate=false

	<bean id="match" class="com.pack.Match" autowire-candidate="false">
	    <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>
	    <constructor-arg index="1" type="java.lang.String" value="1000"/>
	  </bean>
	  <bean id="match1" class="com.pack.Match">
	    <constructor-arg index="0" type="java.lang.String" value="Delhi"/>
	    <constructor-arg index="1" type="java.lang.String" value="2000"/>
	  </bean>
	  <bean id="player" class="com.pack.Player" autowire="constructor">
	    <constructor-arg index="0" type="java.lang.Integer" value="1001"/>
	    <constructor-arg index="1" type="java.lang.String" value="Kohli"/>
	  </bean>

	primary="true"
```

- If same bean is configured multiple times in xml file, which bean class has to be injected, it is defined using primary property
- So which bean we want to use we have to define primary="true"

```
<bean id="match" class="com.pack.Match" primary="true">
    <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>
    <constructor-arg index="1" type="java.lang.String" value="1000"/>
  </bean>
  <bean id="match1" class="com.pack.Match">
    <constructor-arg index="0" type="java.lang.String" value="Delhi"/>
    <constructor-arg index="1" type="java.lang.String" value="2000"/>
  </bean>
  <bean id="player" class="com.pack.Player" autowire="constructor">
    <constructor-arg index="0" type="java.lang.Integer" value="1001"/>
    <constructor-arg index="1" type="java.lang.String" value="Kohli"/>
  </bean>
```

Spring using Annotation

- used instead of configuring bean in xml file

@Configuration - class level annotation, something like xml file, where we configure all bean classes (ie) collection of bean classes

@Bean - used to configure single bean class

```
public class Example1 {
  private String message1;

        public String getMessage1() {
                return message1;
        }

        public void setMessage1(String message1) {
                this.message1 = message1;
        }

}

public class Example2 {
  private String message2;

public String getMessage2() {
        return message2;
}

public void setMessage2(String message2) {
        this.message2 = message2;
}

}

@Configuration
public class ExampleConfiguration {

        @Bean
        @Scope("prototype")
        public Example1 example1() {
                return new Example1();
        }

        @Bean
        public Example2 example2() {
                return new Example2();
        }
}


public class Main2 {
        public static void main(String[] args) {
                ApplicationContext ctx=
                                new AnnotationConfigApplicationContext(ExampleConfiguration.class);
                /*Example1 e1=(Example1)ctx.getBean(Example1.class);
```

```
                e1.setMessage1("Hello world");
                System.out.println(e1.getMessage1());

                Example2 e2=(Example2)ctx.getBean(Example2.class);
                e2.setMessage2("Welcome world");
                System.out.println(e2.getMessage2());*/

                Example1 e1=(Example1)ctx.getBean(Example1.class);
                System.out.println(e1);
                Example1 e2=(Example1)ctx.getBean(Example1.class);
                System.out.println(e2);
        }
}
```

Subject: RE: Reg: Day 16 Notes

Classification: **Confidential**

Handson
1. Create User bean class with id, name, address, salary and inject the values using setter injection, constructor injection and print the output

J2EE - to develop web oriented appl - J2EE Components - Traditional approach
1. JSP - Java Server Pages - used to create dynamic web pages - UI Part
2. Servlets - java prg used to write business logic
3. EJB - Enterprise Java Bean - 3 beans
    1. Session bean - write business logic
    2. Entity bean - persist into db
    3. Message Driven bean - used to send sync and async messages

Problems with traditional approach
1. J2EE applications tend to contain excessive amounts of "plumbing" code -> Thers would always be a high proportion of code that doesn't do anything: JNDI lookup code, Transfer Objects, try/catch blocks to acquire and release JDBC resources. Writing and maintaining such plumbing code proves a major drain on resources that should be focused on the application's business domain.

2. J2EE applications are hard to unit test -> The J2EE APIs, especially, the EJB component model, does not take into account ease of unit testing. It is very difficult to test applications based on EJB and many other J2EE APIs outside an application server.

3. Certain J2EE technologies have failed in performance. EJB 2.x, for instance -> The main offender here is entity beans, which have proven little short of disastrous for productivity.

J2EE Frameworks
1. There are many frameworks which claim to resolve the issues mentioned earlier. For instance, Struts.

2. Struts is a web framework which works on the web tier and helps us achieve MVC and is doing pretty well in the market. However Spring takes over struts in that it is not just a web framework but an application framework.

3. Unlike single-tier frameworks such as Struts or Hibernate, Spring aims to help structure whole applications in a consistent, productive manner. It has modules that offer services for use throughout an application.

4. The essence of Spring is in providing enterprise services to Plain Old Java Objects (POJOs). This is valuable in a J2EE environment.

Spring framework
- open source framework, used to develop both standalone(Java) as well as Web(J2EE) appl. It is a framework so we have to download all jar files and put inside appl

Features
1. Reduce glue code/plumbing work
Spring Framework takes lot of load off the programmer by providing dependencies when required and by using AOP(Aspect Oriented Programming) - define common functionality in single place(bean) and access it anywhere

2. Externalize dependencies - Inversion of Control(IOC) - Dependency Injection
Dependencies are described in a separate file (xml) rather than mixing it with the business logic code itself. This gives a better control over the application.
Loosely coupled appl - inject the values at runtime defined in separate xml file

3. Manage dependencies at a single place
Dependencies can be managed better due to this.

4. Improve testability
Actual code can easily be replaced by a stub for testing purposes.

5. Foster good application design
Since the actual implementation sits behind the interfaces, it fosters good application design.

6. Flexibility
Spring offers integration points with several other frameworks. So, you do not have to write them yourself.

7 modules
1. Spring Core module
- support IOC concept
- Using BeanFactory interface

2. Spring Context module
- supports IOC concept + support internationalization, lifecycle events, integration with EJB, emailing, remoting, scheduling, integration with FreeMarker and Velocity framework
- Using ApplicationContext interface

3. Spring AOP(Aspect Oriented Programming)
- define common functionality in single place(bean) and access it anywhere
eg: security, transaction mgt, logging

4. Spring DAO(Data Access Object) module
- It provides some inbuilt interface to connect with database using Spring itself(JdbcTemplate class)

5. Spring ORM(Object Relational Mapping) module
- Used to integrate Spring with Hibernate framework

6. Spring Webflow module
- Used to integrate Spring with Struts framework

7. Spring MVC module
- used to develop MVC architecture using Spring itself

IOC - Inversion of Control
Dependency Injection
- objects are given their dependencies at runtime in separate xml file
3 types of DI
1. Setter injection - Spring
- dependencies are configured through beans setter methods for their properties using <property> tag
2. Constructor injection - Spring
- dependencies are configured through beans contructors for their properties using <constructor-arg> tag
3. Interface injection - Avalon
- dependencies are configured through interface for their properties

Spring IOC container - 2 types of IOC containers are there to perform DI
1. BeanFactory interface
- org.springframework.beans.factory.* package
- It is lazy loading
- used to create and destroy the bean
- Use XmlBeanFactory(Resource res) class to load the beans defined in xml file
- Object getBean(String name) - used to instantiate the bean and it performs DI

2. ApplicationContext interface
- org.springframework.context.* package
- It is eager loading
- used to create and destroy the bean
- 3 classes
1. ClassPathXmlApplicationContext(String xmlfilename) - loads the xml file from classpath(inside src folder)
2. FileSystemXmlApplicationContext(String xmlfilename) - loads the xml file from file system (ie) c:/ or d:/ or inside project
3. WebXmlApplicationContext(String xmlfilename) - loads the xml file from web.xml

```
public class Main {
	public static void main(String[] args) {
		//Resource res=new ClassPathResource("hello.xml");
		//Resource res=new FileSystemResource("hello.xml");
		//BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file
		//Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI
		//System.out.println(st.getName()+" "+st.getAge());
```

```
                //ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");
                ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
                Student st=(Student)context.getBean("stud");
                System.out.println(st.getName()+" "+st.getAge());
        }
}

public class Student {
    private Integer id;
    private String name;
    private Integer age;
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Integer getAge() {
                return age;
        }
        public void setAge(Integer age) {
                this.age = age;
        }

}


public class Employee {
  private Integer id;
  private String name;
  private Double salary;
  private int age;
public Employee(Integer id, String name, Double salary, int age) {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.age = age;
}
public Employee() {
        super();
        // TODO Auto-generated constructor stub
}
@Override
public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", age=" + age + "]";
}

}


<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springframework.org/schema/beans
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:context=http://www.springframework.org/schema/context
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <bean id="stud"  class="com.pack.Student">
      <property name="id" value="1000"/>
      <property name="name" value="Ram"/>
      <property name="age" value="23"/>
   </bean>
```

```
  <!--  <bean id="emp" class="com.pack.Employee">
      <constructor-arg  name="id" type="java.lang.Integer" value="2000"/>
      <constructor-arg  name="name" type="java.lang.String" value="Sam"/>
      <constructor-arg  name="salary" type="java.lang.Double" value="20000.0"/>
      <constructor-arg  name="age" type="int" value="20"/>
   </bean> -->

    <bean id="emp" class="com.pack.Employee">
      <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>
      <constructor-arg  index="1" type="java.lang.String" value="Sam"/>
      <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>
      <constructor-arg  index="3" type="int" value="20"/>
   </bean>

</beans>


public class Main {
	public static void main(String[] args) {
		//Resource res=new ClassPathResource("hello.xml");
		//Resource res=new FileSystemResource("hello.xml");
		//BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file
		//Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI
		//System.out.println(st.getName()+" "+st.getAge());

		//ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");
		//ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
		//Student st=(Student)context.getBean("stud");
		//System.out.println(st.getName()+" "+st.getAge());

		ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
		Employee e=(Employee)context.getBean("emp");
		System.out.println(e);
	}
}
```

3 programs
1. Bean prg - simple pojo class contains getter and setter methods
2. xml file - used to perform dependency injection
3. main class

Apache Maven
- It is project development tool, instead of downloading the jar files manually, maven will download all the jar files on behalf of the user from internet
- pom.xml(project object model) - we have to provide dependency
- maven will contains 2 repo

1. Remote repository - for first time maven will download all jar files remotely present in internet and put in local repo
2. Local repository - maven will internally create a folder .m2(local repo)

Installation
1. Download maven
https://maven.apache.org/download.cgi

Binary zip archive apache-maven-3.9.5-bin.zip

2. Extract apache maven
3. configure maven in env variable

In Search - Type env - Edit env variables for ur account

1. Under user variable - click New

Variable name: MAVEN_HOME
Variable value: C:\Softwares\apache-maven-3.6.3

2. Select Path - Click Edit - Click New - Paste C:\Softwares\apache-maven-3.6.3\bin

4. Open cmd prompt
>mvn -version

5. >mvn clean
- It will say build failure, used to create local repo .m2

goto c:/users/urname/.m2

Spring project
1. Click File - New - Create Maven Project - Next

Check Create a simple project - Click Next

Group id: com.pack (package name)
ArtifactId: Spring1 (projectname)
Versioning: jar

Click Finish

2. Configure dependency in pom.xml

```
<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>4.2.4.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.2.4.RELEASE</version>
		</dependency>
	</dependencies>
```

3. Right click project - click properties - Copy path from location - click apply and close

4. Open cmd prompt

C:\Users\senthil.kumart>cd C:\Spring\Spring1

C:\Spring\Spring1>mvn clean install
- It will download all jar files and keep inside local repo

5. Right click project - Maven - Update project - check Force update on snapshot - click ok

6. Create bean program inside src/main/java

```
public class Student {
   private Integer id;
   private String name;
   private Integer age;
	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}

}

public class Employee {
  private Integer id;
```

```
  private String name;
  private Double salary;
  private int age;
public Employee(Integer id, String name, Double salary, int age) {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.age = age;
}
public Employee() {
        super();
        // TODO Auto-generated constructor stub
}
@Override
public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", age=" + age + "]";
}

}
```

7. Create xml file in any name inside src or inside project
   - used to configure bean prg inside xml file to perform dependency injection
   - contain root elt called <beans> and each bean class is configured by <bean> tag
   1. id - any name which acts as the object of bean class
   2. class - fully qualified path of bean prg

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springframework.org/schema/beans
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:context=http://www.springframework.org/schema/context
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <bean id="stud"  class="com.pack.Student">
    <property name="id" value="1000"/>
    <property name="name" value="Ram"/>
    <property name="age" value="23"/>
  </bean>

 <!--  <bean id="emp" class="com.pack.Employee">
    <constructor-arg  name="id" type="java.lang.Integer" value="2000"/>
    <constructor-arg  name="name" type="java.lang.String" value="Sam"/>
    <constructor-arg  name="salary" type="java.lang.Double" value="20000.0"/>
    <constructor-arg  name="age" type="int" value="20"/>
  </bean> -->

   <bean id="emp" class="com.pack.Employee">
    <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>
    <constructor-arg  index="1" type="java.lang.String" value="Sam"/>
    <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>
    <constructor-arg  index="3" type="int" value="20"/>
  </bean>

</beans>
```

8. Main class

Resource interface is used to locate the xml file - 2 classes
  1. ClassPathResource class - if xml file present in src folder
  2. FileSystemResource class - if xml file present in file system or project

```
public class Main {
        public static void main(String[] args) {
                //Resource res=new ClassPathResource("hello.xml");
                //Resource res=new FileSystemResource("hello.xml");
                //BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file
                //Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI
                //System.out.println(st.getName()+" "+st.getAge());
```

```
                //ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");
                //ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
                //Student st=(Student)context.getBean("stud");
                //System.out.println(st.getName()+" "+st.getAge());

                ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");
                Employee e=(Employee)context.getBean("emp");
                System.out.println(e);
        }
}
```

Subject: RE: Reg: Day 16 Notes

Classification: **Confidential**

Handson

emp (eno(pk), ename, bdate, title, salary, dno)
proj (pno(pk), pname, budget, dno)
dept (dno(pk), dname, mgreno)
workson (eno(fk), pno(fk), resp, hours)

Questions:
1) Write an SQL query that returns the project number and name for projects with a budget greater than $100,000.
2) Write an SQL query that returns all works on records where hours worked is less than 10 and the responsibility is 'Manager'.
3) Write an SQL query that returns the employees (number and name only) who have a title of 'EE' or 'SA' and make more than $35,000.
4) Write an SQL query that returns the employees (name only) in department 'D1' ordered by decreasing salary.
5) Write an SQL query that returns the departments (all fields) ordered by ascending department name.
6) Write an SQL query that returns the employee name, department name, and employee title.
7) Write an SQL query that returns the project name, hours worked, and project number for all works on records where hours > 10.
8) Write an SQL query that returns the project name, department name, and budget for all projects with a budget < $50,000.
9) Write an SQL query that returns the employee numbers and salaries of all employees in the 'Consulting' department ordered by descending salary.
10) Write an SQL query that returns the employee name, project name, employee title, and hours for all works on records.

Subqueries
- select query inside an another select query
- used to retrieve data based on unknown condition
- Inner query will be executed first and based on output the outer query will be executed
- 2 types of subqueries

1. Single row subquery - inner query returns a single row, then it is evaluated with outer query using =,!=,<,>,<=,>=
2. Multi row subquery - inner query returns a multiple rows, then it is evaluated with outer query using any,all,in, not in, exists, not exists

```
mysql> create table emp(eid int primary key, efirstname varchar(20),elastname varchar(20),ecity varchar(20));
Query OK, 0 rows affected (0.06 sec)

mysql> insert into emp values(1,'Ram','Kumar','Chenna');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp values(2,'Sam','Kumar','Pune');
Query OK, 1 row affected (0.00 sec)

mysql> insert into emp values(3,'Saj','Kumar','Mumbai');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp values(4,'Raj','Kumar','Delhi');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp values(5,'John','Smith','Bangalore');
Query OK, 1 row affected (0.01 sec)

mysql> insert into emp values(6,'Jane','Baker','Hyderabad');
Query OK, 1 row affected (0.01 sec)

mysql> create table dept(did int primary key,dname varchar(20),empid int references emp(eid),dsal float);
Query OK, 0 rows affected (0.07 sec)

mysql> insert into dept values(1000,'HR',1,5000);
```

```
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept values(1001,'Sales',2,10000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept values(1002,'IT',3,5000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept values(1003,'Admin',4,10000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept values(1004,'HR',5,12000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept values(1005,'IT',6,15000);
Query OK, 1 row affected (0.01 sec)

mysql> select * from emp;
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   1 | Ram        | Kumar     | Chenna    |
|   2 | Sam        | Kumar     | Pune      |
|   3 | Saj        | Kumar     | Mumbai    |
|   4 | Raj        | Kumar     | Delhi     |
|   5 | John       | Smith     | Bangalore |
|   6 | Jane       | Baker     | Hyderabad |
+-----+------------+-----------+-----------+
6 rows in set (0.00 sec)

mysql> select * from dept;
+------+-------+-------+-------+
| did  | dname | empid | dsal  |
+------+-------+-------+-------+
| 1000 | HR    |     1 |  5000 |
| 1001 | Sales |     2 | 10000 |
| 1002 | IT    |     3 |  5000 |
| 1003 | Admin |     4 | 10000 |
| 1004 | HR    |     5 | 12000 |
| 1005 | IT    |     6 | 15000 |
+------+-------+-------+-------+
6 rows in set (0.00 sec)

mysql> select * from emp where eid=(select empid from dept where dsal=12000);
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   5 | John       | Smith     | Bangalore |
+-----+------------+-----------+-----------+
1 row in set (0.01 sec)

mysql> select * from emp where eid in (select empid from dept where dsal>10000);
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   5 | John       | Smith     | Bangalore |
|   6 | Jane       | Baker     | Hyderabad |
+-----+------------+-----------+-----------+
2 rows in set (0.00 sec)

mysql> select * from emp where eid = (select empid from dept where dsal>10000);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> select * from emp where eid > any(select empid from dept where dsal<10000);
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   2 | Sam        | Kumar     | Pune      |
|   3 | Saj        | Kumar     | Mumbai    |
|   4 | Raj        | Kumar     | Delhi     |
|   5 | John       | Smith     | Bangalore |
|   6 | Jane       | Baker     | Hyderabad |
```

```
+-----+------------+-----------+-----------+
5 rows in set (0.01 sec)

mysql> select * from emp where eid > all(select empid from dept where dsal<10000);
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   4 | Raj        | Kumar     | Delhi     |
|   5 | John       | Smith     | Bangalore |
|   6 | Jane       | Baker     | Hyderabad |
+-----+------------+-----------+-----------+
3 rows in set (0.00 sec)

mysql> select * from emp where eid = all(select empid from dept where dsal<10000);
Empty set (0.00 sec)

mysql> select * from emp where exists(select * from dept where dsal=16000);
Empty set (0.00 sec)

mysql> select * from emp where not exists(select * from dept where dsal=16000);
+-----+------------+-----------+-----------+
| eid | efirstname | elastname | ecity     |
+-----+------------+-----------+-----------+
|   1 | Ram        | Kumar     | Chenna    |
|   2 | Sam        | Kumar     | Pune      |
|   3 | Saj        | Kumar     | Mumbai    |
|   4 | Raj        | Kumar     | Delhi     |
|   5 | John       | Smith     | Bangalore |
|   6 | Jane       | Baker     | Hyderabad |
+-----+------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Joins
- at the time of displaying we can join the table and get the data

Types
1. Equi join
   - we join the table based on some equality condition

```
mysql> create table dept1(deptid int primary key,dname varchar(20));
Query OK, 0 rows affected (0.13 sec)

mysql> insert into dept1 values(10,'Sales');
Query OK, 1 row affected (0.03 sec)

mysql> insert into dept1 values(20,'Admin');
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept1 values(30,'HR');
Query OK, 1 row affected (0.01 sec)

mysql> insert into dept1 values(40,'IT');
Query OK, 1 row affected (0.01 sec)

mysql> create table emp1(empid int primary key,ename varchar(20),deptid int references dept1(deptid));
Query OK, 0 rows affected (0.11 sec)

mysql> insert into emp1 values(1,'Pat',10);
Query OK, 1 row affected (0.03 sec)

mysql> insert into emp1 values(2,'Sat',20);
Query OK, 1 row affected (0.03 sec)

mysql> insert into emp1 values(3,'Pack',10);
Query OK, 1 row affected (0.03 sec)

mysql> insert into emp1(empid,ename) values(4,'Mack');
Query OK, 1 row affected (0.03 sec)

mysql> select * from dept1;
```

```
+--------+-------+
| deptid | dname |
+--------+-------+
|     10 | Sales |
|     20 | Admin |
|     30 | HR    |
|     40 | IT    |
+--------+-------+
4 rows in set (0.00 sec)

mysql> select * from emp1;
+-------+-------+--------+
| empid | ename | deptid |
+-------+-------+--------+
|     1 | Pat   |     10 |
|     2 | Sat   |     20 |
|     3 | Pack  |     10 |
|     4 | Mack  |   NULL |
+-------+-------+--------+
4 rows in set (0.00 sec)

mysql> select e.ename,d.dname from emp1 e, dept1 d where e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
+-------+-------+
3 rows in set (0.00 sec)

2. Non equi join
       - Join the table based on some non equality conditions like >,<,<=,>=,!=

3. Natural join
      - create an implicit join clause based on common columns in a table

mysql> select * from emp1 natural join dept1;
+--------+-------+-------+-------+
| deptid | empid | ename | dname |
+--------+-------+-------+-------+
|     10 |     1 | Pat   | Sales |
|     20 |     2 | Sat   | Admin |
|     10 |     3 | Pack  | Sales |
+--------+-------+-------+-------+
3 rows in set (0.00 sec)

4. Inner join
     - Similar to Equi join but we have to use "on" clause

mysql> select e.ename,d.dname from emp1 e inner join dept1 d on e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
+-------+-------+
3 rows in set (0.00 sec)

5. Outer join
    - used to display both matched and unmatched data

3 types
1. Left outer join
     - all rows from left table + only matched rows right table

mysql> select e.ename,d.dname from emp1 e left join dept1 d on e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
```

```
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
| Mack  | NULL  |
+-------+-------+
4 rows in set (0.00 sec)
```

2. Right outer join
- all rows from right table + only matched rows from left table

```
mysql> select e.ename,d.dname from emp1 e right join dept1 d on e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
| NULL  | HR    |
| NULL  | IT    |
+-------+-------+
5 rows in set (0.00 sec)
```

3. Full outer join
- used to display both matched and unmatched rows from both table
- In mysql, full outer join is not possible, we use union of left outer join and right outer join

```
mysql> select e.ename,d.dname from emp1 e left join dept1 d on e.deptid=d.deptid
   -> union
   -> select e.ename,d.dname from emp1 e right join dept1 d on e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
| Mack  | NULL  |
| NULL  | HR    |
| NULL  | IT    |
+-------+-------+
6 rows in set (0.01 sec)
```

6. Cross join
- cartesian product of two table

```
>select * from emp1 cross join dept1;
    or
>select * from emp1,dept1;
```

4. TCL - Transaction Control Language
- Transaction is a small unit of work that is performed against the database (ie) while DML operation
- 3 stmt

1. commit - used to commit the transaction performed in the database
2. rollback - undo all previous transaction before committing, only for DML

```
insert;
commit;
insert;
insert;
update;
rollback
```

3. savepoint - used to undo particular transaction

```
insert;
delete
create; - when we call DDL stmt inbetween, then autocommit will take place
insert;
rollback;
```

```
mysql> select * from emp1;
```

```
+-------+-------+--------+
| empid | ename | deptid |
+-------+-------+--------+
|     1 | Pat   |     10 |
|     2 | Sat   |     20 |
|     3 | Pack  |     10 |
|     4 | Mack  |   NULL |
+-------+-------+--------+
4 rows in set (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into emp1 values(5,'Raj',20);
Query OK, 1 row affected (0.02 sec)

mysql> update emp1 set deptid=30 where empid=4;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from emp1;
+-------+-------+--------+
| empid | ename | deptid |
+-------+-------+--------+
|     1 | Pat   |     10 |
|     2 | Sat   |     20 |
|     3 | Pack  |     10 |
|     4 | Mack  |     30 |
|     5 | Raj   |     20 |
+-------+-------+--------+
5 rows in set (0.00 sec)

mysql> insert into emp1 values(6,'Rajj',20);
Query OK, 1 row affected (0.01 sec)

mysql> create table emp2(id int,name varchar(20));
Query OK, 0 rows affected (0.28 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SAVEPOINT initial;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into emp1 values(10,'Taj',30);
Query OK, 1 row affected (0.02 sec)

mysql> SAVEPOINT addition;
Query OK, 0 rows affected (0.00 sec)

mysql> update emp1 set ename='Tajjjj' where empid=10;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from emp1;
+-------+--------+--------+
| empid | ename  | deptid |
+-------+--------+--------+
|     1 | Pat    |     10 |
|     2 | Sat    |     20 |
|     3 | Pack   |     10 |
|     4 | Mack   |     30 |
|     5 | Raj    |     20 |
|     7 | Saj    |     20 |
|    10 | Tajjjj |     30 |
+-------+--------+--------+
7 rows in set (0.00 sec)
```

```
mysql> SAVEPOINT updation;
Query OK, 0 rows affected (0.00 sec)

mysql> delete from emp1 where empid=10;
Query OK, 1 row affected (0.00 sec)

mysql> select * from emp1;
+-------+-------+--------+
| empid | ename | deptid |
+-------+-------+--------+
|     1 | Pat   |     10 |
|     2 | Sat   |     20 |
|     3 | Pack  |     10 |
|     4 | Mack  |     30 |
|     5 | Raj   |     20 |
|     7 | Saj   |     20 |
+-------+-------+--------+
6 rows in set (0.00 sec)

mysql> SAVEPOINT deletion;
Query OK, 0 rows affected (0.00 sec)

mysql> rollback to updation;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from emp1;
+-------+--------+--------+
| empid | ename  | deptid |
+-------+--------+--------+
|     1 | Pat    |     10 |
|     2 | Sat    |     20 |
|     3 | Pack   |     10 |
|     4 | Mack   |     30 |
|     5 | Raj    |     20 |
|     7 | Saj    |     20 |
|    10 | Tajjjj |     30 |
+-------+--------+--------+
7 rows in set (0.00 sec)

mysql> rollback to addition;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from emp1;
+-------+-------+--------+
| empid | ename | deptid |
+-------+-------+--------+
|     1 | Pat   |     10 |
|     2 | Sat   |     20 |
|     3 | Pack  |     10 |
|     4 | Mack  |     30 |
|     5 | Raj   |     20 |
|     7 | Saj   |     20 |
|    10 | Taj   |     30 |
+-------+-------+--------+
7 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.03 sec)
```

5. DCL - Data Control Language
   - GRANT - granting privilege to user
   - REVOKE - removing privilege from user

```
>grant select,update on emp to system;
>revoke select,update on emp to system;
```

schema - anything that is stored into database (ie) table,views, procedure etc

Views
- It is a schema object
- It is called as virtual table because it dosent have its own data, it will always contain data of another table, that table is called base table

- used to execute complex queries

2 types
1. Simple view
2. Complex view

Syntax: create [or replace] view viewname as select stmt;

```
mysql> select e.ename,d.dname from emp1 e,dept1 d where e.deptid=d.deptid;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
| Mack  | HR    |
| Raj   | Admin |
| Saj   | Admin |
| Taj   | HR    |
+-------+-------+
7 rows in set (0.00 sec)

mysql> create view v1 as select e.ename,d.dname from emp1 e,dept1 d where e.deptid=d.deptid;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from v1;
+-------+-------+
| ename | dname |
+-------+-------+
| Pat   | Sales |
| Sat   | Admin |
| Pack  | Sales |
| Mack  | HR    |
| Raj   | Admin |
| Saj   | Admin |
| Taj   | HR    |
+-------+-------+
7 rows in set (0.02 sec)

mysql> show tables;
+--------------------------+
| Tables_in_trainingbatch |
+--------------------------+
| dept                     |
| dept1                    |
| emp                      |
| emp1                     |
| emp2                     |
| v1                       |
+--------------------------+
6 rows in set (0.01 sec)

mysql> show full tables;
+--------------------------+------------+
| Tables_in_trainingbatch | Table_type |
+--------------------------+------------+
| dept                     | BASE TABLE |
| dept1                    | BASE TABLE |
| emp                      | BASE TABLE |
| emp1                     | BASE TABLE |
| emp2                     | BASE TABLE |
| v1                       | VIEW       |
+--------------------------+------------+
6 rows in set (0.02 sec)

mysql> drop table emp1;
Query OK, 0 rows affected (0.05 sec)

mysql> show full tables;
+--------------------------+------------+
| Tables_in_trainingbatch | Table_type |
```

```
+-------------------------+------------+
| dept                    | BASE TABLE |
| dept1                   | BASE TABLE |
| emp                     | BASE TABLE |
| emp2                    | BASE TABLE |
| v1                      | VIEW       |
+-------------------------+------------+
5 rows in set (0.00 sec)

mysql> select * from v1;
ERROR 1356 (HY000): View 'trainingbatch.v1' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

mysql> drop view v1;
Query OK, 0 rows affected (0.02 sec)

mysql> show full tables;
+-------------------------+------------+
| Tables_in_trainingbatch | Table_type |
+-------------------------+------------+
| dept                    | BASE TABLE |
| dept1                   | BASE TABLE |
| emp                     | BASE TABLE |
| emp2                    | BASE TABLE |
+-------------------------+------------+
4 rows in set (0.00 sec)
```

Updatable views

- Certain views are updatable (ie) if we perform any DML operation on views it will affect the base table, but with following condition on select stmt
1. select stmt should not contain distinct,set operator, aggregate function, group by clause, having clause
2. left join, outer join
3. subquery in select list

```
mysql> create view v2 as select * from dept1;
Query OK, 0 rows affected (0.04 sec)

mysql> insert into v2 values(60,'Finance');
Query OK, 1 row affected (0.01 sec)

mysql> select * from dept1;
+--------+----------+
| deptid | dname    |
+--------+----------+
|     10 | Sales    |
|     20 | Admin    |
|     30 | HR       |
|     40 | IT       |
|     50 | Accounts |
|     60 | Finance  |
+--------+----------+
6 rows in set (0.00 sec)
```

with check option in views

- Certain views are updatable, but at the time of updating the view, if we want to check some condition

```
mysql> create view v4 as select * from dept1 where deptid>50 with check option;
Query OK, 0 rows affected (0.03 sec)

mysql> insert into v4 values(70,'Operation');
Query OK, 1 row affected (0.01 sec)

mysql> select * from v4;
+--------+-----------+
| deptid | dname     |
+--------+-----------+
|     60 | Finance   |
|     70 | Operation |
+--------+-----------+
2 rows in set (0.00 sec)

mysql> insert into v4 values(55,'Production');
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into v4 values(45,'Development');
ERROR 1369 (HY000): CHECK OPTION failed 'trainingbatch.v4'
mysql> delete from v4 where deptid=55;
Query OK, 1 row affected (0.03 sec)

mysql> delete from v4 where deptid=10;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from dept1;
+--------+-----------+
| deptid | dname     |
+--------+-----------+
|     10 | Sales     |
|     20 | Admin     |
|     30 | HR        |
|     40 | IT        |
|     50 | Accounts  |
|     60 | Finance   |
|     70 | Operation |
+--------+-----------+
7 rows in set (0.00 sec)
```

Rename view
>rename table v3 to v4;

Dropping view
>drop view v3;

Alter the view
1. using create or replace
2. First drop the view and recreate the view once again

Index
- It is a schema object
- used for faster retrieval of data, if ur table contains huge number of data
- Index will be automatically create for primary key and unique key which is called as BTree index

Syntax: create index indexname on tablename(columnname,....)

```
mysql> select * from dept1;
+--------+-----------+
| deptid | dname     |
+--------+-----------+
|     10 | Sales     |
|     20 | Admin     |
|     30 | HR        |
|     40 | IT        |
|     50 | Accounts  |
|     60 | Finance   |
|     70 | Operation |
+--------+-----------+
7 rows in set (0.00 sec)

mysql> create index idx1 on dept1(dname);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show indexes from dept1;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
Index_comment | Visible | Expression |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| dept1 |          0 | PRIMARY  |            1 | deptid      | A         |           7 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| dept1 |          1 | idx1     |            1 | dname       | A         |           7 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.05 sec)

mysql> drop index idx1 on dept1;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show indexes from dept1;
+-------+------------+---------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+---------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| dept1 |          0 | PRIMARY |            1 | deptid      | A         |           7 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-------+------------+---------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.00 sec)
```

Subject: RE: Reg: Day 15 Notes

Classification: **Confidential**

Create the table WORLDCITY with the following data.

City	Country	Continent	Latitude	NorthSouth	Longitude	EastWest
Athens	Greece	Europe	37.59	N	23.44	E
Atlanta	United States	North America	33.45	N	84.23	W
Dallas	United States	North America	32.47	N	96.47	W
Nashville	United States	North America	36.09	N	86.46	W
Victoria	Canada	North America	48.25	N	123.21	W
Peterborough	Canada	North America	44.18	N	79.18	W
Vancouver	Canada	North America	49.18	N	123.04	W
Toledo	United States	North America	41.39	N	83.82	W
Warsaw	Poland	Europe	52.15	N	21.00	E
Lima	Peru	South America	12.03	S	77.03	W
Rio De Janeiro	Brazil	South America	22.43	S	43.13	W
Santiago	Chile	South America	33.27	S	70.40	W
Bogota	Colombia	South America	04.36	N	74.05	W
Buenos Aires	Argentina	South America	34.36	S	58.28	W
Quito	Ecuador	South America	00.13	S	78.30	W
Caracas	Venezuela	South America	10.30	N	66.56	W
Madras	India	Asia	28.36	N	77.12	E
Bombay	India	Asia	18.58	N	72.50	E
Manchester	England	Europe	51.30	N	0.0	null
Moscow	Russia	Europe	55.45	N	37.35	E
Paris	France	Europe	48.52	N	2.20	E
Shenyang	China	Asia	41.48	N	123.27	E
Cairo	Egypt	Africa	30.03	N	31.15	E
Tripoli	Lybia	Africa	32.54	N	13.11	E
Beijing	China	Asia	39.56	N	116.24	E
Rome	Italy	Europe	41.54	N	12.29	E
Tokyo	Japan	Asia	35.42	N	139.46	E
Sydney	Australia	Australia	33.52	S	151.13	E
Sparta	Greece	Europe	37.05	N	22.27	E
Madrid	Spain	Europe	40.24	N	3.41	W

Complete the following exercise:

1. For all the different countries contained in the WORLDCITY table, display their names and the continent in which they are located. Make sure that no country name is duplicated.

1. Write an SQL query to display the list of the city and country for all the cities that begin with letter R.

2. Write an SQL query to display the list of the city and country for all the cities that end with letter A.
3. Write an SQL query to display the list of the city and country for all the cities that begin with letter M and have exactly six letters in them.
4. Write an SQL query to display the list of the city and country for all the cities that contain an A as the second letter.

Complete the following exercise:

a) Create a following tables

CUSTOMER Table Schema:

COLUMN NAME	DESCRIPTION	DATATYPE	SIZE	CONSTRAINTS
CUSTID	Customer Id	Number		Primary Key
FNAME	First Name	Character	30	
LNAME	Last Name	Character	30	
ADDRESS	Customer Address	Character	50	
PHONENO	Phone	Number		Not Null
CITY	City	Character	20	
COUNTRY	Country	Character	20	
DATEFIRSTPURCHASED	Date of the first purchase by the customer	Date		
SUPPLIERID	Supplier Information	Number		Foreign Key

Data for CUSTOMER table:

CUSTID	FNAME	LNAME	ADDRESS	PHONENO	CITY	COUNTRY	DATEFIRSTPURCHASED	SUPPLIERID
1001	Das	Jeyaseelan	119, park Avenue, II street,	9841093428	Coimbatore	India	10-jan-2004	1
2001	Gopi	Govindraj	241, I floor, Kamaraj street, Madippakkam	9444124590	Chennai	India	25-mar-2005	4
1201	Dilip	Kishore	43, II Avenue, Anna Nagar	9997234534	Bangalore	India	20-aug-2004	2
1300	Aanand	Chowdhury	42/1 sector 1, II Street	9841054348	Bangalore	India	15-may-2005	2
1220	Chandra	Nagarajan	83, lal bagh	98410672356	Bangalore	India	12-feb-2006	4
1221	Abhishek	Kumar	13,kishori park,	94447623901	Chennai	India	15-may-2004	1
1320	Nikhil	Pandit	218, alwaanya street	94448923091	Salem	India	21-apr-2006	3
1222	Meenu	Monica	C11, church road	98410563421	Trichy	India	30-aug-2004	1
1225	Pavan	Kumar	128/A, North Mada Street	99934782103	maduari	India	18-aug-2004	4

SUPPLIER Table Schema:

COLUMN NAME	DESCRIPTION	DATATYPE	SIZE	CONSTRAINTS
SUPPLIERID	Supplier Id	Number		Primary Key
SNAME	Supplier Name	Character	30	
SCITY	Supplier City	Character	30	
SPHONE	Supplier Phone	Number		Not Null
EMAIL	Email id of Supplier	Character	50	Unique

Data for SUPPLIER table:

SUPPLIERID	SNAME	SCITY	SPHONE	EMAIL
1	Dilip	Chennai	8999900000	dilip@abc.co.in
2	Tarun	Madurai	8999911111	tarun@xyz.com
3	Naresh	Coimbatore	8999922222	g.naresh@xyzl.com
4	Ganesan	Trichy	8999933333	Ganesan_83@ijk.com

ORDERS Table Schema:

COLUMN NAME	DESCRIPTION	DATATYPE	SIZE	CONSTRAINTS
ORDERID	Order Number	Number		Primary Key
ORDERDATE	Date of Order	Date		

CUSTID	Customer Identity	Number		
QUANTITY	Quantity Ordered	Number		Check (Quantity >0)
ITEMID	Item Code	Number		Foreign Key

Data for ORDERS Table:

ORDERID	ORDERDATE	CUSTID	QUANTITY	ITEMID
1	12-jan-2004	1001	30	25
2	6-may-2005	1202	38	24
3	16-dec-2006	1220	10	22
4	21-may-2004	1233	12	21

ITEMS Table Schema:

COLUMN NAME	DESCRIPTION	DATATYPE	SIZE	CONSTRAINTS
ITEMID	Item Code	Number		Primary Key
ITEMNAME	Item Name	Character	35	Not Null
SUPPLIERID	Supplier Code	Number		Foreign Key
MINQTY	Minimum Qty that can be ordered	Number		Not Null
MAXQTY	Maximum Qty that can be ordered	Number		Not Null
Price	Price per unit	Number(5,2)		

Data for ITEMS Table:

ITEMID	ITEMNAME	SUPPLIERID	MINQTY	MAXQTY	Price
20	Pears Soap	4	7	20	30.00
21	V.V.D. Coconut oil 200 ml	2	8	15	79.00
22	Ponds powder 400g	3	6	25	106.00
23	Reynolds pen- blue	1	10	30	15.00
24	Reynolds pen- black	1	10	30	16.00
25	Mysore sandal soap	4	7	25	25.00
26	Fair & lovely cream- 50g	3	5	15	55.00
27	Rexono deo spary	2	5	20	100.00
28	Dove soap	4	7	15	85.00

b) Write SQL queries to

i. Display all customers from Chennai.
ii. Display the details of all customers who purchased from the supplier 2.
iii. Display CUSTID, FNAME, LNAME of all customers whose purchase date is after January 2005.
iv. Display the details of all suppliers who are from location Coimbatore.
v. Display the details of all suppliers whose name stars with G.
vi. Display the details of all customers, who do not have the alphabet e in their LNAME.
vii. Display the details of the entire customer whose first date of purchase is in 2006 and arrange it in descending order.
viii. Display the details of all the orders where the quantity is less than 35.
ix. Display the details of the items supplied by supplier 4.
x. Display the details of all items where SUPPLIERID is 3 and the MINQTY is greater than 7 order by ITEMID.

SQL Statements
1. DDL
2. DML
3. DQL - Data Query Language

1. select - used to select the data from table

> select * from customers; - select all rows and columns from table

> select custid,name from customers; - select all rows but only specific column

> select * from customers where custid>5; - select all columns but specific rows

>select custid,name from customers where custid>5; - select only specific column and specific rows

>select order.orderid, supplier.supname from order,supplier where order.orderid=supplier.orderid; - select from multiple table

>select o.orderid, s.supname from order o,supplier s where o.orderid=s.orderid; - create an alias for table to select data from multiple tables

>select distinct city from customers; - To avoid duplicates values

>select * from suppliers where city='Chennai' and type='PC manufacture';

>select * from suppliers where city='Chennai' or type='PC manufacture';

IS NULL, IS NOT NULL - to check null values

>select * from emp where dept is null;

>select * from emp where dept is not null;

Pattern Matching Operator - LIKE, NOT LIKE
- used to fetch data based on some pattern - applied only for varchar
- 2 wildcard characters
  % - anything
  _ - single value

>select * from suppliers where supname like 'R%'; - select all suppliers whose name should start with R

>select * from suppliers where supname like '%bob%';
>select * from suppliers where supname not like 'T%';
>select * from suppliers where supname like 'Sm_th';
>select * from orders where orderid like '1000_';

supname
Hello%
HelloHo
HelloBye
Hello%

>select * from suppliers where supname like 'Hello%'; - all 4 rows

If we want to include %,_ at time of pattern matching then we have to use escape sequence

>select * from suppliers where supname like 'Hello!%' ESCAPE '!'; - Return 2 rows

Gana
Girl
Giri
G%

>select * from suppliers where supname like 'G%';

>select * from suppliers where supname like 'G&%' ESCAPE '&';

Hello
Hello%%
Hi%

>select * from suppliers where supname like 'H%';

>select * from suppliers where supname like 'H%!%' ESCAPE '!';

>select * from suppliers where supname like 'H%\%';

IN, NOT IN - used of using multiple or operator

>select * from suppliers where supname='HCL' or supname='CTS' or supname='TCS';

>select * from suppliers where supname in('HCL','CTS',TCS');
>select * from orders where orderid not in(1000,2000,3000);

BETWEEN, NOT BETWEEN - Range operator - includes the ranges

>select * from suppliers where supid between 1000 and 2000;

>select * from suppliers where supid>=1000 and supid<=2000;

Set operator - UNION, UNION ALL, INTERSECT, MINUS

>select supid from supplier
union
select supid from order;
- select both supid from supplier and order table, if there is any duplication it will be considered only once

>select supid from supplier
union all
select supid from order;
- select both supid from supplier and order table, and print everything with duplication

>select supid from supplier
intersect
select supid from order;
- select only common supid from supplier and order table

>select supid from supplier
minus
select supid from order;
- remove supid from supplier that is present in order table

SQL Functions
1. Numeric function - apply these function to numbers

1. abs(n) - return absolute value
>select abs(9); 9
>select abs(-9); 9

2. ceil(n) - largest number >= n
>select ceil(9.5); 10
>select ceil(-9.5); -9

3. floor(n) - nearest number <= n
>select floor(9.5); 9
>select floor(-9.5); -10

4. mod(m,n) - mod(7,2) = 1
5. powe(m,n) - power(3,2) = 9
6. sqrt(n)
7. exp(n)
8. log(n)
9. sin(n)
10. cos(n)
11. tan(n)
12. round(n) - round to nearest decimal value

>select round(125.07); 125
>select round(125.78); 126
>select round(125.67,1); 125.7
>select round(125.11,1); 125.1
>select round(125.67,2); 125.67
>select round(125.678,2); 125.68
>select round(125.67,-1); 130 120 125 130
-1 represents 10's, if the value is greater than mid value it returns higher value otherwise it returns lower value
>select round(222.12,-1); 220 220 225 230
>select round(222.12,-2); 200 200 250 300
-2 represents 100's, if the value is greater than mid value it returns higher value otherwise it returns lower value
>select round(673.12,-2); 700 600 650 700
>select round(3567.12,-3); 4000 3000 3500 4000
-3 represents 1000's, if the value is greater than mid value it returns higher value otherwise it returns lower value

13. truncate(n)

>select truncate(125.13); error
>select truncate(125.13,1); 125.1
>select truncate(125.67,1); 125.6
>select truncate(125.678,2); 125.67
>select truncate(125.67,-1); 120
-1 represents 10's, if the value is greater or lesser than mid value it always returns only lower value

>select truncate(435.12,-2); 400

2. Character function - apply only on varchar
1. ascii(n) - return ascii value of left most char
>select ascii('a'); 97
>select ascii('bad'); 98

2. char_length(n)
character_length(n)
length(n)
- used to find length of string

>select char_length('hello'); 5
>select character_length('hello'); 5
>select length('hello'); 5

3. concat()/space operator - concatenate two or more string

>select concat('The answer is',24); The answer is 24
>select concat('The answer is', 10+10); The answer is 20
>select concat('The answer is', '10+10'); The answer is 10+10
>select concat('The answer is', null); null
>select 'The answer is' 24; The answer is 24

4. concat_ws() - concatenate two or more expr and adds a separator between each concatenated expr

>select concat_ws('+','a','b','c'); a+b+c
>select concat_ws('xyz','A','B','C'); AxyzBxyzC
>select concat_ws(null,'1','2'); null

5. field() - return the posiiton of a value in a list of values

>select field('b','a','b','c','d','e','f'); 2
>select field(15,10,20,15,34); 3
>select field('c','a','b'); 0
>select field(null,'a','b'); null

6. find_in_set() - return the posiiton of a value in a comma delimited string

>select find_in_set('b','a,b,c,d,e,f'); 2
>select find_in_set(2,'3,4,1,5,2'); 5

7. format() - used to format number with comma and rounding to nearest decimal places

>select format(123453.6789,2); 1,23,453.68

8. instr() - return location of substring in a string

>select instr('hello','h'); 1
>select instr('hello','e'); 2
>select instr('hello','i'); 0

9. lcase()
lower()
- convert string to lowercase

10. left() - extract leftmost char from a string

>select left('hello',1); h
>select left('hello',4); hell
>select left('hello',20); hello

11. right() - extract rightmost char from a string

>select right('hello',1); o
>select right('hello',4); ello
>select right('hello',20); hello

12. locate() - return the location of first occurence of substring in a string

>select locate('H','Hello'); 1

```
>select locate('l','Hello');  3
>select locate('l','Hello',3); 4  - from 3rd positin, first occurence of l
>select locate('l','heelo',3); 4
>select locate('e','Technology internet'); 2
>select locate('e','technology internet',3); 15
>select locate('z','technology internet',3); 0
```

13. lpad(char1,n,char2) - char1 is left padded with char2 of length n

```
>select lpad('world',10,'hello');  helloworld
>select lpad('world',11,'hello');  hellohworld
>select lpad('world',12,'hello');  helloheworld
>select lpad('world',13,'hello');  hellohelworld
>select lpad('world',9,'hello');  hellworld
>select lpad('world',4,'hello');  worl
```

14. rpad(char1,n,char2) - char1 is right padded with char2 of length n

```
>select rpad('world',10,'hello');  worldhello
>select rpad('world',11,'hello');  worldhelloh
>select rpad('world',12,'hello');  worldhellohe
>select rpad('world',13,'hello');  worldhellohel
>select rpad('world',9,'hello');   worldhell
>select rpad('world',4,'hello');   worl
```

15. ltrim() - removes all spaces from left side

```
>select ltrim(' Hello '); Hello_
>select ltrim(' Hello world ');  Hello world_
```

16. rtrim() - removes all spaces from right side

```
>select rtrim(' Hello '); _Hello
>select rtrim(' Hello world ');  _Hello world
```

17. trim() - trims both leading and trailing space

```
>select trim(leading from ' hello '); hello_
>select trim(trailing from ' hello '); _hello
>select trim(both from ' hello '); hello
>select trim(leading '0' from '00012340'); 12340
>select trim(trailing '1' from '111hello111'); 111hello
```

18. ucase()
upper()
- convert string to uppercase

19. mid() - used to extract substring from a string

```
>select mid('technology',5,2);   no
>select mid('technology',1,4);   tech
>select mid('technology',-7,4);  hnol
>select mid('technology',-3,3);  ogy
```

20. substr() - used to extract substring from a string

```
>select substr('technology.com',5);  nology.com
>select substr('technology.com',1,4);  tech
>select substr('technology.com',-3,3);  com
>select substr('technology.com' from 1 for 4); tech
```

21. substring() - used to extract substring from a string

```
>select substring('technology.com',5);  nology.com
>select substring('technology.com',1,4);  tech
>select substring('technology.com',-3,3);  com
>select substring('technology.com' from 1 for 4); tech
```

22. substring_index() - return substring from string before number of occurences of delimiter

```
>select substring_index('www.technology.com','.',1); www
>select substring_index('www.technology.com','.',2); www.technology
>select substring_index('www.technology.com','.',-1); com

23. strcmp() - compare 2 string

>select strcmp('hello','hello');  0
>select strcmp('cat','hello');  -1
>select strcmp('hello','cat');  1

24. space() - return a string with specified number of space

>select space(3);  '   '

25. reverse() - reverse the string

>select reverse('abcd');  dcba

26. replace()

>select replace('abc abc','a','B');  Bbc Bbc

3. Aggregate function/Group function
     - used to group any value
      - It will ignore the null value at a time of grouping
      - sum(),avg(), max(), min(), count(), stddev(), median()
      - For date datatype, we can apply only  max(), min(), count()

Employee table
empid(int)  name(varchar)  gender(varchar) salary(double) dob(date) - consider we have totally 30 rows

What we provide in select list that only will be coming as column name in output

>select count(*) from employee;

count(*)
-------
30

>select count(gender) from employee;

count(gender)
-------------
30

>select count(distinct gender) from employee;

count(distinct gender)
-------------
2

Create an Alias for the column
>select count(*) as "Number of Employees" from employee;

Number of Employees
-------------------
30

>select count(*) "Number of Employees" from employee;

Number of Employees
-------------------
30

>select count(*) NumberofEmployees from employee;

NumberofEmployees
-------------------
30

>select sum(salary) as "Total Salary" from employee;
```

>select max(dob) as DOB from employee;
>select min(dob) DOB from employee;
>select stddev(salary) from employee;
>select median(salary) from employee;

order by clause
- used for sorting purpose only at time of displaying
- It should be always present in the last of select stmt
select - from - where - group by - having - order by
- By default sorted in ascending order
- column index represent the column from select list not from table
- If we sort on two or more columns, then by default it will be sorted only on first column, in the first column if there is any repition for those state column alone, the city will be sorted in desc order

>select supplier_city,supplier_state from supplier where supname='IBM' order by supplier_city;

>select supplier_city,supplier_state from supplier where supname='IBM' order by supplier_city asc;

>select supplier_city,supplier_state from supplier where supname='IBM' order by supplier_city desc;

Supplier table
sup_id supname supplier_city supplier_state address

>select supplier_city,supplier_state from supplier where supname='IBM' order by 2 desc; //column index represent the column from select list not from table

Supplier
supplier_city supplier_state
Chennai Tamilnadu
Allepey Kerela
Coimbatore Tamilnadu
Mumbai Maharastra
Pune Maharastra

>select supplier_city,supplier_state from supplier where supname='IBM' order by supplier_state asc, supplier_city desc;

Supplier
supplier_state supplier_city
Kerela Allepey
Maharastra Pune
Maharastra Mumbai
Tamilnadu Coimbatore
Tamilnadu Chennai

group by clause
- used to group the values in a table based on some aggregate function

Employee table
empid ename dept salary
1 A HR 4000
2 B IT 6000
3 C HR 6000
4 D Admin 5000
5 E IT 10000
6 F Admin 6000
7 G HR 5000

We want to calculate the total sum of salary
>select sum(salary) from employee;

We want to calculate the total sum of salary in each dept
>select dept,sum(salary) from employee group by dept;

dept sum(salary)
HR 15000
IT 16000
Admin 11000

having clause
- used to exclude the result from groupby clause

```
>select dept,sum(salary) from employee group by dept having sum(salary)>14000
order by dept asc;

dept   sum(salary)
HR      15000
IT      16000
```

Subject: RE: Reg: Day 14 Notes

Classification: **Confidential**

Complete the following assignment:

a) Create the table PROGRAMMER with the given information using SQL CREATE TABLE command:

Attribute Name	Description/Data Type/Constraint
EmpNo	Employee's unique ID. Max. 5 characters should be numeric
ProjId	Project in which programmer participates. Max. 5 characters should be varchar
LastName	Surname of employee. Max. 30 characters. Required.
FirstName	Employee's first name. Max. 30 characters.
HireDate	Date on which employee was hired. Date data type.
Language	Programming language used by programmer. Max. 15 characters
TaskNo	Number of the task associated with the project. Numeric column, max. 2 digits
Privilege	Type of privilege given to programmer. Max. 25 characters.

b) Insert the following data into the PROGRAMMER table:

EmpNo	LastName	FirstName	HireDate	ProjId	Language	TaskNo	Privilege
201	Gupta	Saurav	1/1/95	NPR	VB	52	Secret
390	Ghosh	Pinky	1/05/93	KCW	Java	11	Top Secret
789	Agarwal	Praveen	08/31/98	Rnc	VB	11	Secret
134	Chaudhury	Supriyo	07/15/95	TIPPS	C++	52	Secret
896	Jha	Ranjit	06/15/97	KCW	Java	10	Top Secret
345	John	Peter	11/15/99	TIPPS	Java	52	
563	Anderson	Andy	08/15/94	NITTS	C++	89	confidential

c) Create a table WEATHER with following data:

City	State	High	Low
Calcutta	West Bengal	105	90
Trivandrum	Kerala	101	92
Mumbai	Maharashtra	88	69
Bangalore	Karnataka	77	60
New Delhi		80	72

d) Create a table BOOKS with the following data

BookId	Title	TopicId	Publisher Name	Placeof Publication	Price	PurchaseDate	ShelfNo
8293	DBMS	DB1	Prentice Hall	Mumbai	255	1/1/95	S11
5645	DBMS	DB1	Pearson Education	Mumbai	655	1/05/93	S12
6565	C	C1	TMH	Mumbai	840	08/31/98	S66
6567	C++	Cplus1	ABC Publishers	Delhi	300	07/15/95	S77
4576	JAVA	JAVA1	Guru Govind Publications	Delhi	500	06/15/97	S87
3433	OOPS	OOPS1	Dave Publishers	Pune	600	11/15/99	S56
4655	SAD	SAD1	Sajan Publications	Cochin	700	08/15/94	S76

Note:

- ❑ The primary keys and other keys could be assumed if not mentioned.
- ❑ When inserting values try using all the 3 different ways to insert data into the table

e) Write SQL queries to:

i. Saurav Gupta is assigned a different project with id NITTS and he would work with C++ now. Update this change in the PROGRAMMER table.

i. The books on DBMS are shifted to shelf with number S10. Please update this detail in BOOKS table.

ii. Supriyo Chaudhury has resigned his job. Incorporate this change in the table PROGRAMMER.

iii. A new column to state the nature of the climate with either of the value (rainy, cloudy, sunny, snow) is to be added in the WEATHER Table.

iv. Delete the table WEATHER from database.

Enum
- It is type of class that mainly stores constants or fixed set of values
- It implicitly abstract class, by default it is final so we cant inherit and it can implemented by a interface

```
interface A {
	void add();
}
enum Day1 implements A {
  SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY;

	@Override
	public void add() {

	}
}
/*enum Day2 extends Day1{

}*/
enum Day {
   SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
}
public class Main {
	public static void main(String[] args)  {
    Day d1=Day.TUESDAY;
    System.out.println(d1);  //TUESDAY

    Day d2=d1;
    System.out.println(d2); //TUESDAY
    //Day d3=10; //error

    Day d3=Day.valueOf("MONDAY");
    System.out.println(d3); //MONDAY
    //Day d4=Day.valueOf("monday");
    //System.out.println(d4); //IllegalArgumentException
    System.out.println(d1==d2); //true
    System.out.println(d1.equals(d2)); //true
    System.out.println(Day.FRIDAY);  //FRIDAY

    for(Day d:Day.values()) {
        System.out.println(d.name()+" "+d.ordinal());
    }
	}
}
```

Rules
1. enum values are declared first, then only we can define anything

```
enum Day {
	//int a=10;  //error
  SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY;
	int s=10; //correct
}
```

2. If we want to define anything after enum values then it should end with semicolon

3. A non final enum method can be overridden by any enum value

```
enum Day {
```

```
	SUNDAY {
		public void printTemp() {
			System.out.println("55");
		}
	},
	MONDAY,
	TUESDAY,
	WEDNESDAY,
	THURSDAY,
	FRIDAY,
	SATURDAY;
	private String temperature;

	public void printTemp() {
		System.out.println("65");
	}
}
public class Main {
	public static void main(String[] args)  {
    Day d1=Day.TUESDAY;
    d1.printTemp(); //65
    Day.SUNDAY.printTemp(); //55
	}
}
```

4. If we have abstract method then it should be implemented by all enum value

```
enum Day {
	SUNDAY {
		public void printTemp() {
			System.out.println("55");
		}
	},
	MONDAY {
		@Override
		public void printTemp() {
			System.out.println("50");
		}
	},
	TUESDAY {
		@Override
		public void printTemp() {
			System.out.println("45");
		}
	},
	WEDNESDAY {
		@Override
		public void printTemp() {
			System.out.println("40");
		}
	},
	THURSDAY {
		@Override
		public void printTemp() {
			System.out.println("35");
		}
	},
	FRIDAY {
		@Override
		public void printTemp() {
			System.out.println("25");
		}
	},
	SATURDAY {
		@Override
		public void printTemp() {
			System.out.println("36");
		}
	};
	private String temperature;
```

```
        public abstract void printTemp();
}
public class Main {
        public static void main(String[] args)  {
    Day d1=Day.TUESDAY;
    d1.printTemp(); //45
    Day.SUNDAY.printTemp(); //55
        }
}
```

5. Constructor is implicitly private, called once in the beginning to create enum value

```
enum Day {
        SUNDAY("High") {
                public void printTemp() {
                        System.out.println("55");
                }
        },
        MONDAY("Low") {
                @Override
                public void printTemp() {
                        System.out.println("50");
                }
        },
        TUESDAY("High") {
                @Override
                public void printTemp() {
                        System.out.println("45");
                }
        },
        WEDNESDAY("Moderate") {
                @Override
                public void printTemp() {
                        System.out.println("40");
                }
        },
        THURSDAY("Low") {
                @Override
                public void printTemp() {
                        System.out.println("35");
                }
        },
        FRIDAY("High") {
                @Override
                public void printTemp() {
                        System.out.println("25");
                }
        },
        SATURDAY("Low") {
                @Override
                public void printTemp() {
                        System.out.println("36");
                }
        };
        String temperature;

        /*private*/ Day(String temperature){
                this.temperature=temperature;
        }

        public abstract void printTemp();
}
public class Main {
        public static void main(String[] args)  {
    Day d1=Day.TUESDAY;
    d1.printTemp(); //45
    Day.SUNDAY.printTemp(); //55

    for(Day d:Day.values()) {
          System.out.println(d+" "+d.temperature);
    }
```

```
        }
}

enum Marks {
        ENGLISH(67),
        MATHS(100),
        SCIENCE(99),
        SOCIAL(100),
        LANGUAGE(89);
        int mark;
  Marks(int mark){
        this.mark=mark;
  }
}
public class Main1 {

        public static void main(String[] args) {
                for(Marks m:Marks.values()) {
                        System.out.println(m+" "+m.mark);
                }
        }

}
```

6. Enum can also used in switch case

```
public class Main {
        public static void main(String[] args)  {
   Day1 d1=Day1.FRIDAY;
   switch(d1) {
   case SUNDAY:
         System.out.println("High");
         break;
   case MONDAY:
         System.out.println("Low");
         break;
   case TUESDAY:
         System.out.println("Moderate");
         break;
   default:
         System.out.println("No comments");
         break;
   }
        }
}
```

File processing system
- we store all data inside the files

Drawbacks
1. Data redundancy - duplication of data
2. Data isolation - same data are scattered across multiple machines
3. Only particular format of data can be stored inside the file
4. No Security
5. No backup and recovery of data
6. Integrity constraints are buried under programming language
7. Only single user can access the file

Database system
- set of logical related data - database
- set of programs to access those data - DBMS/RDBMS

DBMS	RDBMS
1. No relationship between tables we have only primary key	1. We can relate two tables using foreign key concepts
2. Data is stored in flat file system	2. Data is stored inside the tables
3. Only single user can access the data at a time	3. Multiple user can access the data

eg: MS Access, foxpro eg: MySQL, Oracle, SQL Server, PostgresSQL, DB2, Sybase

Data Models
- semantic representation of data

1. Hiearchial model
2. Network model
3. ER Model - Entity Relationship diagram
4. Relational model - represent the data in the form of tables
   Tables will contains rows(records/tuples) and columns(attributes/fields)
5. Semistructured model - represent data in xml file

DBMS/RDBMS
- set a programs to access the data - 2 types
1. SQL - Structured Query Language
2. PLSQL - Procedural Language Structured Query Language

SQL
- It is a english like syntax to process the data in database

Types of keys in SQL
1. Super key - any combination of columns that uniquely fetch the value from database

Consider we have Employee table - empid,ename,address,salary,dept

ename,address,dept - super keys
empid,salary - candidate key (minimal of super key)
empid - primary key (anyone of candidate key is selected as primary key)
salary - alternate key

Integrity Constraints - 5 types
1. Entity integrity constraint
   - Entity means table - constraints that applied directly on the table
   - 2 types
  1. Primary key - no null value, no duplication
  2. Unique key - no duplication but it will allow null value

2. Domain Integrity constraint
   - Domain is permitted value for the column
   - constraint that is applied to value for the column
   - 2 types
     1. NOT NULL
     2. CHECK

3. Referential Integrity Constraint
   - We can refer one table with another table using foreign key concept
   - One tables PK only will be acting as FK in another table, so first we have to create PK table then only we have to create foreign table
   - Foreign key contains duplicate values and also contain null values
   - FK column should contain only the values that are present in PK column
   - whenever we have one to many,many to one, many to many relationship
   - First we have to delete PK table then we can delete FK table

one person places many orders - one to many

Person table

pid(pk)	pname
1	A
2	B
3	C
4	D
5	E

Orders table

ordid(pk)	ordername	pid(FK)
100	Coffee	1
101	Tea	2
102	Chips	3
103	Juice	4
105	Biscuits	5

106 Cakes 1
107 Cola

4. Default constraint
- By default if we didnt provide any value for the column, irrespective of the datatype always it will contain null value
- But we dont want to have null value, instead we want to provide some default value, then we can use default constraint

5. NULL constraint

Composite primary key
- If table contains more than one column as primary key where its combination should be unique

A(PK)	B(PK)	C	
1	1	1	11
2	1		21
1	4	3	14
1	2	4	12

Note
1. By default if we didnt provide any value for the column, irrespective of the datatype always it will contain null value
2. SQL is a case sensitive - tablename, columnname is not case sensitivity, but the values stored inside the columns are case sensitive
'ram'!='Ram'

SQL Datatypes
1. Numeric datatype - store any numbers +ve or -ve
- numeric(p,q) p=total width, q=precision after decimal point
numeric(5,2) - 123.45
- numeric(p)
numeric(5) - 12345
- int
- integer
- tinyint -128 to 127
- smallint -32768 to 32767
- mediumint -8388608 to 8388607
- bigint -9223372036854775808 to 9223372036854775807
- decimal(p,q)/fixed(p,q)
- float
- double

2. Character datatype
- char(n) - fixed length of char
char(10)='ram'
- varchar(n) - variable length of char
varchar(10)='ram'

3. Date datatype
- date - only date(yyyy-MM-dd)
- datetime - both date and time (yyyy-MM-dd hh:mm:ss)
- timestamp - both date and time(yyyyMMddhhmmss)

4. LOB datatype - Large Object datatype
- BLOB(Binary Large Object) - used to store images, audio, video
- CLOB(Character Large Object) - used to store very long string

SQL Operators
1. +,-,*,/,<,>,<=,>=,!=
2. ALL
3. ANY
4. AND
5. OR
6. IN, NOT IN
7. LIKE, NOT LIKE - Pattern matching operator
8. BETWEEN, NOT BETWEEN- - Range operator
9. IS NULL, IS NOT NULL
10. EXISTS, NOT EXISTS
11. UNION, UNION ALL, INTERSECT, MINUS - Set operator

mysql> show databases; - to view all databases

mysql> create database javabatch; - create a new db
Query OK, 1 row affected (0.03 sec)

```
mysql> use javabatch;  - use the db
Database changed
mysql> show tables;  - to view all tables in db
Empty set (0.02 sec)

mysql> drop database batch2;  - delete the db
Query OK, 13 rows affected (0.24 sec)
```

SQL Statement
1. DDL statement - Data Defination Language - 4 stmt
- create, alter, drop, truncate

a. create - create a new table in db

```
mysql> create table customer(custid integer,name varchar(20),age int);
Query OK, 0 rows affected (0.07 sec)

mysql> desc customer;   - to view structure of table
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| custid | int         | YES  |     | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)


mysql> create table customer3(sno int unique,custid integer primary key AUTO_INCREMENT,name varchar(20),age int);
Query OK, 0 rows affected (0.07 sec)

mysql> create table customer4(sno int,custid integer,name varchar(20),age int,constraint cust_pk primary key(sno,custid));
Query OK, 0 rows affected (0.05 sec)

mysql> create table customer5(sno int unique,custid integer primary key AUTO_INCREMENT,name varchar(20) NOT NULL,age int default 25);
Query OK, 0 rows affected (0.06 sec)

mysql> create table department(deptid int primary key,dname varchar(20));
Query OK, 0 rows affected (0.06 sec)

mysql> create table employee(empid int primary key,ename varchar(20) NOT NULL, age int, did int references department(deptid));
Query OK, 0 rows affected (0.07 sec)
```

- Copy both data and structure from one table and create an another table

```
mysql> create table customer6 as select * from customer5;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create table customer7 as select custid,name from customer5;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- Copy only structure from one table and create an another table

```
mysql> create table customer8 as select * from customer5 where 1=20;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

b. alter

1. To rename table name

```
mysql> alter table customer8 rename to cust8;
Query OK, 0 rows affected (0.04 sec)
```

2. To rename column name

```
mysql> alter table cust8 rename column age to custage;
Query OK, 0 rows affected (0.01 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0

3. To add new column to exiting table

mysql> desc cust8;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| sno     | int         | YES  |     | NULL    |       |
| custid  | int         | NO   |     | 0       |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| custage | int         | YES  |     | 25      |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> alter table cust8 add address varchar(40);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table cust8 add city varchar(40),add state varchar(40);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc cust8;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| sno     | int         | YES  |     | NULL    |       |
| custid  | int         | NO   |     | 0       |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| custage | int         | YES  |     | 25      |       |
| address | varchar(40) | YES  |     | NULL    |       |
| city    | varchar(40) | YES  |     | NULL    |       |
| state   | varchar(40) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> alter table cust8 add email varchar(40) after custage;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc cust8;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| sno     | int         | YES  |     | NULL    |       |
| custid  | int         | NO   |     | 0       |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| custage | int         | YES  |     | 25      |       |
| email   | varchar(40) | YES  |     | NULL    |       |
| address | varchar(40) | YES  |     | NULL    |       |
| city    | varchar(40) | YES  |     | NULL    |       |
| state   | varchar(40) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

4. Modify only datatype of existing column

mysql> alter table cust8 modify address varchar(50) NOT NULL;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table cust8 modify city varchar(50) NOT NULL,modify state varchar(50) NOT NULL;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc cust8;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| sno     | int         | YES  |     | NULL    |       |
```

```
| custid  | int         | NO   |     | 0       |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| custage | int         | YES  |     | 25      |       |
| email   | varchar(40) | YES  |     | NULL    |       |
| address | varchar(50) | NO   |     | NULL    |       |
| city    | varchar(50) | NO   |     | NULL    |       |
| state   | varchar(50) | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

5. To drop existing column in table

```
mysql> alter table cust8 drop column email;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6. To add constraint to the table after creating

```
mysql> alter table customer add constraint cu_pk primary key(custid);
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

7. To enable/disable primary key or unique key

```
mysql> alter table customer disable keys;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> alter table customer enable keys;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

To enable/disable foreign key

```
SET FOREIGN_KEY_CHECKS=0;
SET FOREIGN_KEY_CHECKS=1;
```

c. drop

- drop the data as well as the table structure

```
mysql> drop table customer;
Query OK, 0 rows affected (0.04 sec)

mysql> drop table customer1,customer2;
Query OK, 0 rows affected (0.05 sec)

mysql> drop table if exists cust8;
Query OK, 0 rows affected (0.03 sec)
```

d. truncate

- used to drop the data but the structure will be maintained but we cannot rollback

```
>truncate table customer4;
```

2. DML - Data Manipulation Language - 3 stmt

- insert, update, delete

a. insert - used to insert new row into the table

- insert all columns

```
mysql> insert into customer6 values(1,100,'Ram',24);
Query OK, 1 row affected (0.01 sec)

mysql> insert into customer6 values(2,101,'Sam',25);
Query OK, 1 row affected (0.01 sec)
```

- insert only into particular columns

```
mysql> insert into customer6(sno,custid,name) values(3,103,'Raj');
Query OK, 1 row affected (0.01 sec)

mysql> select * from customer6;
+------+--------+------+------+
| sno  | custid | name | age  |
```

```
+------+--------+------+------+
|   1 |   100 | Ram  |   24 |
|   2 |   101 | Sam  |   25 |
|   3 |   103 | Raj  |   25 |
+------+--------+------+------+
3 rows in set (0.01 sec)
```

- insert data from another table, make sure both table should have same number of col and same column datatype

```
mysql> select * from customer5;
Empty set (0.00 sec)

mysql> insert into customer5 select * from customer6;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from customer5;
+------+--------+------+------+
| sno  | custid | name | age  |
+------+--------+------+------+
|   1 |   100 | Ram  |   24 |
|   2 |   101 | Sam  |   25 |
|   3 |   103 | Raj  |   25 |
+------+--------+------+------+
3 rows in set (0.00 sec)
```

- with single insert query we want to insert multiple rows

```
mysql> insert into customer6(sno,custid,name,age) values
    -> (4,104,'Tam',34),(5,105,'Tim',25),(6,106,'Jim',27);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

b. update - to update value of existing row

```
mysql> update customer6 set age=29 where custid=104;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update customer6 set age=26,name='Jimmy' where custid=106;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

c. delete - used to delete the data from table but the structure will be maintained we can rollback

```
mysql> delete from customer6 where sno=6;
Query OK, 1 row affected (0.01 sec)

mysql> delete from customer6;
Query OK, 5 rows affected (0.01 sec)
```

Subject: RE: Reg: Day 13 Notes

Classification: **Confidential**

Handson

1.Display Date

Given a date in the form of string, write a program to convert the given string to date .
Include a class UserMainCode with a static method displayDate which accepts a string. In this method display the given string in date format yyyy-MM-dd. The return type is void.

Create a Class Main which would be used to accepts a string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.
Output consists of Date.

Refer sample output for formatting specifications.

Sample Input 1:
May 1, 2016
Sample Output 1:
2016-05-01

Sample Input 2:
March 21, 2016
Sample Output 2:
2016-03-21

2.Extract Date and time

Write a program to extract date and time from the input string which is in yyyy-MM-dd HH:mm:ss date format.

Include a class UserMainCode with a static method displayDateTime which accepts a string. In this method display date and time in the format as given in sample input and output . The return type is void.

Create a Class Main which would be used to accept string and call the static method present in UserMainCode.

Input and Output Format:
Input consists of a string.
Output should be in date format

Refer sample output for formatting specifications.

Sample Input :

Enter String in this format(yyyy-MM-DD HH:mm:ss)

2016-07-14 09:00:02
Sample Output :

07/14/2016, 9:00:02

3.Day Of The Year

Given a date, write a program to display day of the year.
Include a class UserMainCode with a static method displayDay which accepts a date. In this method, display day in the format as given in sample input and output . The return type is void.?
?Input date Format is yyyy-MM-dd
Create a Class Main which would be used to accept date and call the static method present in UserMainCode.?
?
Input and Output Format:?
Input consists of a string.
Refer sample output for formatting specifications.

Sample Input :
2013-03-23

Sample Output :
Day of year: 82

4.Name Of the Day

Given a date in the date format, write a program to get the day of the corresponding date .
Include a class UserMainCode with a static method displayDay which accepts a date. In this method display the day of given date . The return type is void.

Create a Class Main which would be used to accept a date and call the static method present in UserMainCode.
?Input date Format is yyyy-MM-dd
Input and Output Format:

Input consists of a date.
Refer sample output for formatting specifications.

Sample Input 1:
2011-10-21
Sample Output 1:
Friday

Sample Input 2:
2011-07-11
Sample Output 2:
Monday

5.Difference between dates in month

Given a method with two date strings in yyyy-mm-dd format as input. Write code to find the difference between two dates in months.

Include a class UserMainCode with a static method getMonthDifference which accepts two date strings as input.

The return type of the output is an integer which returns the diffenece between two dates in months.

Create a class Main which would get the input and call the static method getMonthDifference present in the UserMainCode.

Input and Output Format:
Input consists of two date strings.
Format of date : yyyy-mm-dd.

Output is an integer.
Refer sample output for formatting specifications.

Sample Input 1:
2012-03-01
2012-04-16
Sample Output 1:
1

Sample Input 2:
2011-03-01
2012-04-16
Sample Output 2:
13

Before JDK1.8, Date class present in java.util.* package

1. Date class - print date and time

2. Calendar class - abstract class, used to extract useful info and date and time

3. GregorianCalendar class - concrete implementation of Calendar class, used to extract useful info and date and time

4. DateFormat class - abstract class, used for formating(convert date to string) and parsing(converting string to date), we can display date and time in different format - SHORT,LONG,MEDIUM,FULL - present in java.text.* pkg

5. SimpleDateFormat class - concrete implementation of DateFormat class, used for formating(convert date to string) and parsing(converting string to date), we can display date and time in different format -present in java.text.* pkg

```
public class Main {
	public static void main(String[] args)  {
	    Date d1=new Date();  //print current date and time
	    System.out.println(d1); //Fri Sep 29 13:03:05 IST 2023
	    Date d2=new Date(10000); //print date and time from Jan 1st 1970
	    System.out.println(d2);  //Thu Jan 01 05:30:10 IST 1970

	    Calendar c=Calendar.getInstance(); //current date and time
	    System.out.println(c.get(Calendar.YEAR));
	    System.out.println(c.get(Calendar.MONTH));
```

```
                GregorianCalendar gc=new GregorianCalendar(); //current date and time
                System.out.println(gc.get(Calendar.YEAR));
                System.out.println(gc.get(Calendar.MONTH));
                GregorianCalendar gc1=new GregorianCalendar(2000,10,20,13,23,24);
                System.out.println(gc1.get(Calendar.HOUR));
                System.out.println(gc1.get(Calendar.MINUTE));

                DateFormat df=DateFormat.getInstance(); //default short format
                String s=df.format(d1);
                System.out.println(s);
                DateFormat df1=DateFormat.getDateInstance(DateFormat.LONG);
                String s1=df1.format(d1);
                System.out.println(s1);
                DateFormat df2=DateFormat.getTimeInstance(DateFormat.MEDIUM);
                String s2=df2.format(d1);
                System.out.println(s2);
                DateFormat df3=DateFormat.getDateTimeInstance(DateFormat.FULL,DateFormat.FULL); //default short format
                String s3=df3.format(d1);
                System.out.println(s3);

                String s4="10/03/2023";
                SimpleDateFormat sdf=new SimpleDateFormat("dd/MM/yyyy");
                try {
                        Date d3=sdf.parse(s4);
                        System.out.println(d3); //Fri Sep 29 13:23:04 IST 2023
                        SimpleDateFormat sdf1=new SimpleDateFormat("yyyy-MM-dd");
                        System.out.println(sdf1.format(d3));
                }
                catch(ParseException e) {
                         System.out.println(e);
                }
        }
}
```

Date API
- Available from JDK1.8 onwards
- present in java.time.* package
- Date is immutable class where lower version of Date is not immuntable

class and interfaces
1. LocalDate class - print only date - yyyy-MM-dd
2. TemporalAdjuster interface - used to get extra info from date and time
3. LocalTime class - print only time
4. LocalDateTime class - print both date and time
5. Period class - used to find difference between 2 dates
6. Duration class - used to find difference between 2 time
7. DateTimeFormatter interface - used to display date and time in different format

```
public class Main {
        public static void main(String[] args)  {
            LocalDate l1=LocalDate.now();  //current date
            System.out.println(l1); //2023-09-29

            LocalDate l2=LocalDate.of(2000, 10,20); //create our own date
            System.out.println(l2); //2000-10-20

            LocalDate l3=LocalDate.parse("2001-04-23"); //convert string to date
            System.out.println(l3); //2001-04-23

            LocalDate l4=l3.plusMonths(2);
            System.out.println(l4);
            LocalDate l5=l4.plus(4, ChronoUnit.YEARS);
            System.out.println(l5);

            LocalDate l6=l5.minusDays(100);
            System.out.println(l6);
            LocalDate l7=l6.minus(5, ChronoUnit.WEEKS);
            System.out.println(l7);

            DayOfWeek d1=LocalDate.parse("2023-09-29").getDayOfWeek();
            System.out.println(d1);  //Friday
```

```
int m=LocalDate.parse("2023-09-29").getDayOfMonth();
System.out.println(m);  //29
int y=LocalDate.parse("2023-09-29").getDayOfYear();
System.out.println(y); //272
int mon=LocalDate.parse("2023-09-29").getMonthValue();
System.out.println(mon); //9
System.out.println(l7.isLeapYear());

boolean b1=LocalDate.parse("2023-09-29").isAfter(LocalDate.parse("2023-09-30"));
System.out.println(b1);  //false
boolean b2=LocalDate.parse("2023-09-29").isBefore(LocalDate.parse("2023-09-30"));
System.out.println(b1);  //true
boolean b3=LocalDate.parse("2023-09-29").isEqual(LocalDate.parse("2023-09-30"));
System.out.println(b1);  //false

LocalDate l8=LocalDate.now().with(TemporalAdjusters.firstDayOfMonth());
System.out.println(l8);  //2023-09-01
LocalDate l9=LocalDate.now().with(TemporalAdjusters.lastDayOfMonth());
System.out.println(l9);
LocalDate l10=LocalDate.now().with(TemporalAdjusters.firstDayOfNextYear());
System.out.println(l10);
LocalDate l11=LocalDate.now().with(TemporalAdjusters.next(DayOfWeek.SATURDAY));
System.out.println(l11);  //2023-09-30
LocalDate l12=LocalDate.now().with(TemporalAdjusters.previous(DayOfWeek.THURSDAY));
System.out.println(l12); //2023-09-28

LocalTime t1=LocalTime.now();
System.out.println(t1);
LocalTime t2=LocalTime.now(ZoneId.of("America/Chicago"));
System.out.println(t2);
LocalTime t3=LocalTime.of(9, 39);
System.out.println(t3);
LocalTime t4=LocalTime.parse("09:30");
System.out.println(t4);

LocalTime t5=t4.plusMinutes(23);
System.out.println(t5);
LocalTime t6=t5.plus(4, ChronoUnit.HOURS);
System.out.println(t6);
LocalTime t7=t6.minusNanos(20000);
System.out.println(t7);
LocalTime t8=t7.minus(35, ChronoUnit.SECONDS);
System.out.println(t8);
System.out.println(LocalTime.MAX);
System.out.println(LocalTime.MIN);
System.out.println(LocalTime.MIDNIGHT);

LocalDateTime ld1=LocalDateTime.now();
System.out.println(ld1); //2023-09-29T13:53:24
LocalDateTime ld2=LocalDateTime.of(LocalDate.now(), LocalTime.now());
System.out.println(ld2);
LocalDateTime ld3=LocalDateTime.parse("2023-09-29T13:53:24");
System.out.println(ld3);  //2023-09-29T13:53:24
System.out.println(ld3.toLocalDate());
System.out.println(ld3.toLocalTime());
System.out.println(LocalDateTime.MAX);
System.out.println(LocalDateTime.MIN);

//convert JDK1.5 Date to JDK1.8 LocalDate
Date d11=new Date();
LocalDateTime ld4=LocalDateTime.ofInstant(d11.toInstant(), ZoneId.systemDefault());
System.out.println(ld4);

//convert Calendar to JDK1.8 LocalDate
Calendar cal=Calendar.getInstance();
LocalDateTime ld5=LocalDateTime.ofInstant(cal.toInstant(), ZoneId.systemDefault());
System.out.println(ld5);

LocalDate l13=LocalDate.now();
LocalDate l14=LocalDate.of(2023, 05, 20);
int diff1=Period.between(l13, l14).getDays();
```

```
        System.out.println(diff1);

        int diff2=Period.between(l13, l14).getMonths();
        System.out.println(diff2);

        long diff3=ChronoUnit.YEARS.between(l14, l2);
        System.out.println(diff3);

        LocalTime t9=LocalTime.now();
        LocalTime t10=LocalTime.of(9, 30);
        long diff4=Duration.between(t9,t10).getSeconds();
        System.out.println(diff4);
        long diff5=Duration.between(t9,t10).toHours();
        System.out.println(diff5);
        long diff6=Duration.between(t9,t10).toMinutes();
        System.out.println(diff6);
        long diff7=ChronoUnit.NANOS.between(t9,t10);
        System.out.println(diff7);

        LocalDateTime ld6=LocalDateTime.now();
        String s1=ld6.format(DateTimeFormatter.ISO_DATE_TIME);
        System.out.println(s1);
        String s2=ld6.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"));
        System.out.println(s2);
        String s3=ld6.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT));
        System.out.println(s3);
        String s4=ld6.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.LONG));
        System.out.println(s4);
        String s5=ld6.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM,FormatStyle.FULL));
       System.out.println(s5);
    }
}
```

Optional class
- Object will contain memory reference or null reference, so if we access anything using null reference we get NullPointerException
- To avoid NPE we have to write code to do the null check, so to avoid unpredictable NPE we can go for Optional class
- present in java.util.* package

Methods
1. static Optional empty() - return empty optional
2. static Optional of(T...t) - return an non empty optional, if value is null it return NPE
3. static Optional ofNullable(T...t) - return an non empty optional, if value is null it return empty optional
4. Object get() - return original value from Optional class
5. void ifPresent(Consumer) - if value is present it invokes related consumer
6. boolean isPresent() - return ture if value is present otherwise false
7. Object orElse(Object) - return the value if present otherwise return other value
8. Object orElseGet(Supplier) - return the value if present otherwise it invoke another logic and return the result
9. Object orElseThrow(Supplier) - return the value if present otherwise it will throw an exception

```
public class Main {
    public static void main(String[] args) throws Throwable  {
        Optional o1=Optional.empty();
        System.out.println(o1); //Optional.empty
        Optional o2=Optional.of("John");
        System.out.println(o2); //Optional[John]
        //Optional o3=Optional.of(null);
        //System.out.println(o3);  //NPE

        Optional o4=Optional.ofNullable("Jim");
        System.out.println(o4); //Optional[Jim]
        Optional o5=Optional.ofNullable(null);
        System.out.println(o5);  //Optional.empty

        Optional o6=Optional.of("John");
        System.out.println(o6); //Optional[John]
        System.out.println(o6.get()); //John
        o6.ifPresent(System.out::println); //John
        System.out.println(o6.isPresent()); //true
        Optional o7=Optional.empty();
        System.out.println(o7.isPresent());  //false
        o7.ifPresent(System.out::println);  //no output
```

```
        System.out.println(o7.orElse("Johnny"));  //Johnny
        System.out.println(o7.orElseGet(()->"Peter"));  //Peter
        System.out.println(o7.orElseThrow(NullPointerException::new));
    }
}
```

3. Default and static methods in interface
- Till JDK1.7, interface will contain only abstract method and public static final variables
- From JDK1.8, apart from abstract method, interface can also contain default and static method
- Without affecting the implemented classes, if we want to add any new methods inside the interface then we can go for default methods
- If my functionality is noway related with object(ie) static method, instead of defining inside the class we can define inside the interface
- From JDK1.9, the interface will contain private and private static methods, if we have common functionality inside default method and to avoid duplicate code, we can define those common functionality inside private method and access inside default method whenever needed

```
interface A {
    default void show() {
        System.out.println("Inside A's show");
    }
}
interface B {
    default void show() {
        System.out.println("Inside B's show");
    }
}
class Sample implements A,B {

    @Override
    public void show() {
        //A.super.show();
        B.super.show();
    }

}
public class Main {
    public static void main(String[] args) throws Throwable  {
        Sample s=new Sample();
        s.show();
    }
}
```

Interface member access
1. Accessible from default and private methods within interface
constant variable = yes
abstract method=yes
another default method =yes
private method=yes
static method=yes
private static method=yes

2. Accessible from static methods within interface
constant variable = yes
abstract method=NO
another default method =NO
private method=NO
static method=yes
private static method=yes

3. Accessible from instance methods by implementing interface
constant variable = yes
abstract method=yes
another default method =yes
private method=no
static method=yes
private static method=no

4. Accessible outside interface without implementing interface
constant variable = yes
abstract method=no
another default method =no
private method=no
static method=yes

```
    private static method=no

interface MyInterface {
        //Till JDK1.7
        /*public static final*/ int CONSTANT=0;
        /*public abstract*/ int abstractMethod();

        //From JDK1.8
        /*public*/ default int defaultMethod() {
                abstractMethod();
                privateMethod();
                staticMethod();
                privateStaticMethod();
                return CONSTANT;
        }
        /*public*/ static int staticMethod() {
                privateStaticMethod();
                return CONSTANT;
        }

        //From JDK1.9
        private int privateMethod() {
                abstractMethod();
                defaultMethod();
                staticMethod();
                privateStaticMethod();
                return CONSTANT;
        }
        private static int privateStaticMethod() {

        }
}
class Example implements MyInterface {

        @Override
        public int abstractMethod() {
                defaultMethod();
                MyInterface.staticMethod();
                return CONSTANT;
        }

}
class Example1 {
        public int instanceMethod() {
                MyInterface.staticMethod();
                return MyInterface.CONSTANT;
        }
}
public class Main {
        public static void main(String[] args) throws Throwable  {

        }
}

var keyword
   - Available from JDK10
   - used to infer datatype at runtime
      var a=10;
   - used only inside constructor or methods or loops or compound block
Rules
1. var cant declare without initial value
2. var can be declared in first line and initialized in second line
3. var cannot be initialized with null value without a type
4. var is not permitted in multiple variable declaration
5. var cannot be used to initialize an array
6. var is reserved type name but not reserved keyword, so we can use var as an identifier except for class, interface ,enum
7. var in lambda expr - you cant mix var and non-var  parameters

interface MyInterface {
        void add(int a,int b);
}
```

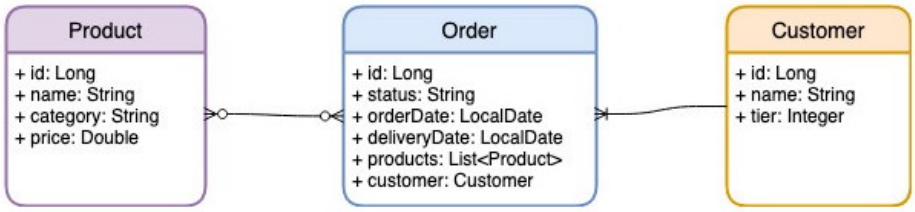
```
class Var {
	Var() {
		var var="var";  //correct
	}
	public void var() { //correct
		Var var=new Var(); //correct
	}
}
/*class var{  //error

}
interface var { //error

}
enum var {  //error

}*/
public class Main {
	Main(){
		var a1="hello";
	}
	{
		var a2=10;
		a2=11;
	}
	public static void main(String[] args) {
	    var size=1;
	    for(var i=0;i<10;i++) {

	    }
	    //var x;  //error
	    var x
	        =3.14f;
	    var x1=23.34;
	    x1=x;
	    var y
	        ="hello";
	    //var z=null; //error
	    var z=(String)null;
	    var z1="hello";
	    z1=null;

	    int a=5,b=6;
	   // var a1=5,b1=9; //error

	   // var a1[]= {1,2,4}; //error

	    MyInterface m1=(var a1,var b1)->System.out.println(a1+b1);  //correct
	    MyInterface m2=(int a1,var b1)->System.out.println(a1+b1);  //error
	}
}
```

Subject: RE: Reg: Day 12 Notes

Classification: **Confidential**

Stream API Handson:

Consider given data model,



Data Model

Use static initializer to create ProductList, OrderList and CustomerList which contains sample Product, Order and Customer

Solve the following questions using Stream API

1. Obtain a list of products belongs to category “Books” with price > 100
2. Obtain a list of order with products belong to category “Baby”
3. Obtain a list of product with category = “Toys” and then apply 10% discount
4. Get the cheapest products of “Books” category
5. Get the 3 most recent placed order
6. Obtain a collection of statistic figures (i.e. sum, average, max, min, count) for all products of category “Books”

Streams API
- contains collection of objects from some sources(Array,List,Set) and process them sequentially
- Collection framework also stores collection of objects and we can manipulate the object, but streams are only used for processing the data and we cant manipulate the data
- present in java.util.stream.* package
- 2 types of streams - finite(fixed number of values) and infinite(unlimited) stream
- 2 types of operation

1. Intermediate operation - return stream itself - optional operation - we can chain multiple intermediate operation
- filter(), map(), flatMap(), sorted(), peek(), distinct(), limit(), skip()

2. Terminal operation - It will traverse into the newly generated stream and return single value - mandatory operation - we cant chain terminal operation
- toArray(), forEach(), count(), min(), max(), reduce(), collect(),anyMatch(), allMatch(), noneMatch(), findFirst()

3 steps
Creation of Stream - Intermediate operation - Terminal Operation
- If we perform terminal operation then that stream is completely closed, so when we perform any other operation on closed stream then we will get IllegalStateException

1. Creation of Stream - Finite stream - 2 ways
1. stream() - used to generate a stream from some source (ie) array or list or set

```
String s[]=new String[]{"one","two","three"};
Stream<String> s1=s.stream();

List<String> l1=new ArrayList<>();
l1.add("Ram");
l1.add("Sam");
l1.add("Raj");
Stream<String> s2=l1.stream();
```

2. of() - used to create our own stream

```
Stream<Integer> st= Stream.of(1,2,3,4,5);
```

filter() - intermediate - used to filter the data based on some condition

reduce() - terminal - used to group the data into single value
- Optional reduce(BinaryOperator)
- Integer reduce(int initialvalue, BinaryOperator)
- Stream reduce(int initialvalue, BinaryOperator b1, BinaryOperator b2)

```
public class Main {
        public static void main(String[] args) {
            List<String> l1=new ArrayList<>();
            l1.add("Ram");
```

```
        l1.add("Sam");
        l1.add("Raj");
        l1.add("Tam");
        l1.add("Tim");
        l1.add("Ram");
        l1.add("John");
        l1.add("Jack");
        System.out.println(l1.size()); //8

        Stream<String> s1=l1.stream();
        Stream<String> s2=s1.distinct();
        long c=s2.count();
        System.out.println(c); //7

        long c1=l1.stream().distinct().count();
        System.out.println(c1); //7,

        boolean b1=l1.stream().distinct().anyMatch((e)->e.startsWith("R"));
        System.out.println(b1);  //true

        boolean b2=l1.stream().distinct().allMatch((e)->e.startsWith("R"));
        System.out.println(b2);  //false

        boolean b3=l1.stream().distinct().noneMatch((e)->e.startsWith("Z"));
        System.out.println(b3);  //true

        List<Student> l2=new ArrayList<>();
        l2.add(new Student(23,"PK"));
        l2.add(new Student(26,"KK"));
        l2.add(new Student(23,"MK"));
        l2.add(new Student(21,"SK"));
        l2.add(new Student(40,"RK"));
        l2.add(new Student(30,"BK"));
        l2.add(new Student(29,"DK"));
        l2.add(new Student(28,"GK"));
        l2.add(new Student(33,"TK"));

        Stream<Student> st=l2.stream().filter((a1)->a1.getId()>25);
        st.forEach(System.out::println);

        Optional opt=Stream.of(3,5,6).reduce((a,b)->a*b);
        System.out.println(opt.get());  //90

        Integer i=Stream.of(3,5,6).reduce(2, (a,b)->a*b);
        System.out.println(i);  //180

        Optional<String> opt1=Stream.of("lion","ape","tiger").min((c11,c21)->c11.length()-c21.length());
        System.out.println(opt1.get());  //ape
    }
}
```

map() - intermediate
- takes one Function functional interface as an argument and returns a stream consisting of result generated by applying the passed function to each element
- used for data transformation

collect() - terminal - used to collect the data as list or set

Collectors.toList()
Collectors.toSet()
Collectors.toMap(Function f1,Function f2)
Collectors.joining()
Collectors.counting()
Collectors.toCollection(Supplier s)
Collectors.partitioningBy() - used to split the list into 2 parts based on true and false
- partitioningBy(predicate)
- partitioningBy(predicate,Collector)

Collectors.groupingBy() - group the stream of elements under some condition
- groupingBy(Function)
- groupingBy(Function,Collector)
- groupingBy(Function,Comparator,Collector)

flatMap() - intermediate
- takes one Function functional interface as an argument and returns a new stream and that stream is copied to another stream which will return the value
- used for data transformation + flatenning

```
public class Main {
	public static void main(String[] args) {
	  Integer a[]=new Integer[] {1,2,3,4,5};
	  List<Integer> l1=Arrays.asList(a);

	  //JDK10
	  //List<Integer> l2=List.of(1,2,3,4,5); //create immutable list

	  List<Integer> l2=l1.stream().map((e)->e*3).collect(Collectors.toList());
	  l2.forEach(System.out::println); //3,6,9,12,15

	  List<Integer> l3=l1.stream().flatMap((e1)->Stream.of(e1*2)).collect(Collectors.toList());
	   l3.forEach(System.out::println);//2,4,6,8,10

	  String s1=Stream.of("one","two","three").collect(Collectors.joining("-"));
	  System.out.println(s1);  //one-two-three

	  long c=Stream.of("one","two","three").collect(Collectors.counting());
	  System.out.println(c);  //3

	 List<String> l4=Stream.of("lions","tigers","bears","toads","toads")
	    .filter((s)->s.startsWith("t")).collect(Collectors.toList());
	  l4.forEach(System.out::println); //tigers,toads,toads

	  Set<String> l5=Stream.of("lions","tigers","bears","toads","toads")
	                .filter((s)->s.startsWith("t")).collect(Collectors.toSet());
	            l5.forEach(System.out::println); //tigers,toads

	             TreeSet<String> ts=Stream.of("lions","tigers","bears","toads","toads","tadpole")
	    .filter((s)->s.startsWith("t")).collect(Collectors.toCollection(TreeSet::new));
	    System.out.println(ts); //[tadpole,tigers,toads]

	  Map<String,Integer>  m1=Stream.of("lions","tigers","bears","toads")
	     .collect(Collectors.toMap(k1->k1, String::length));
	  m1.forEach((k,v)->System.out.println("Key = "+k+" Value = "+v));

	  Map<Boolean,List<String>> m2=Stream.of("lions","tigers","bears","toads","toads","tadpole")
	      .collect(Collectors.partitioningBy((a1)->a1.length()<=5));
	  System.out.println(m2);

	  Map<Boolean,Set<String>> m3=Stream.of("lions","tigers","bears","toads","toads","tadpole")
	                .collect(Collectors.partitioningBy((a1)->a1.length()<=5,Collectors.toSet()));
	            System.out.println(m3);

	           Map<Integer,List<String>> m4=Stream.of("lions","tigers","bears","lions","ape")
	                .collect(Collectors.groupingBy((e)->e.length()));
	           System.out.println(m4);

	           Map<Integer,Set<String>> m5=Stream.of("lions","tigers","bears","lions","ape")
	                .collect(Collectors.groupingBy((e)->e.length(),Collectors.toSet()));
	           System.out.println(m5);

	           TreeMap<Integer,Set<String>> m6=Stream.of("lions","tigers","bears","lions","ape")
	                .collect(Collectors.groupingBy((e)->e.length(),TreeMap::new,Collectors.toSet()));
	           System.out.println(m6);

	}
}
```

sorted() - intermediate - used to sort the elt

```
public class Main {
	public static void main(String[] args) {
	  List<String> l1=List.of("9","A","z","1","B","4","e","f");
```

```
        List<String> l21=l1.stream().sorted().collect(Collectors.toList()); //asc order
        System.out.println(l21);
        List<String> l3=l1.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList()); //desc order
        System.out.println(l3);

        List<Student> l2=new ArrayList<>();
         l2.add(new Student(23,"PK"));
         l2.add(new Student(26,"KK"));
         l2.add(new Student(23,"MK"));
         l2.add(new Student(21,"SK"));
         l2.add(new Student(40,"RK"));
         l2.add(new Student(30,"BK"));
         l2.add(new Student(29,"DK"));
         l2.add(new Student(28,"GK"));
         l2.add(new Student(33,"TK"));

         //comparaingInt(), comparaingDouble(), comparaingLong()
         List<Student> l4=l2.stream().sorted(Comparator.comparingInt(Student::getId))
                                 .collect(Collectors.toList());
         l4.forEach(System.out::println);
         System.out.println();
         List<Student> l5=l2.stream().sorted(Comparator.comparingInt(Student::getId).reversed())
                 .collect(Collectors.toList());
      l5.forEach(System.out::println);
      System.out.println();
      List<Student> l6=l2.stream().sorted(Comparator.comparing(Student::getName))
                 .collect(Collectors.toList());
       l6.forEach(System.out::println);
       System.out.println();
       List<Student> l7=l2.stream().sorted(Comparator.comparing(Student::getName).reversed())
                 .collect(Collectors.toList());
        l7.forEach(System.out::println);
          }
}
```

Other ways to create stream
1. builder() - create finite stream
2. generate() - create infinite stream
3. iterate() - create infinite stream

Create streams based on primitive datatype
1. IntStream - create stream of int values
2. DoubleStream - create stream of double values
3. LongStream - create stream of long values

From JDK1.9
takeWhile() - if stream does not match the predicate, it will discard the rest of stream
dropWhile() - if stream does not match the predicate, it will print the rest of stream

```
public class Main {
        public static void main(String[] args) {
          Stream<String> s1=Stream.<String>builder().add("Ram").add("Sam").add("Raj").build();
          s1.forEach(System.out::println);

          Stream<String> s2=Stream.generate(()->"hello").limit(5);
          s2.forEach(System.out::println);

          Stream<Integer> s3=Stream.iterate(2, (i)->i*2).skip(3).limit(5);
          s3.forEach(System.out::println);

          IntStream i1=IntStream.range(1,6);  //start to end-1
          i1.forEach(System.out::println);  //1 2 3 4 5

          IntStream i2=IntStream.rangeClosed(1,6);  //start to end
          i2.forEach(System.out::println);  //1 2 3 4 5 6

          IntStream i3="abcd".chars();
          i3.forEach(System.out::println);  //97 98 99 100

          Random r=new Random();
          DoubleStream d1=r.doubles(5);
```

```
        d1.forEach(System.out::println);

        List<Student> l2=new ArrayList<>();
         l2.add(new Student(23,"PK"));
         l2.add(new Student(26,"KK"));
         l2.add(new Student(23,"MK"));
         l2.add(new Student(21,"SK"));
         l2.add(new Student(40,"RK"));
         l2.add(new Student(30,"BK"));
         l2.add(new Student(29,"DK"));
         l2.add(new Student(28,"GK"));
         l2.add(new Student(33,"TK"));

         IntStream i5=l2.stream().mapToInt(Student::getId);
         i5.forEach(System.out::println);

        OptionalInt op=l2.stream().mapToInt(Student::getId).max();
        System.out.println(op.getAsInt());  //40

        OptionalDouble op1=l2.stream().mapToDouble(Student::getId).average();
        System.out.println(op1.getAsDouble());

        //IntSummaryStatistics,DoubleSummaryStatistics, LongSummaryStatistics
        IntSummaryStatistics in=l2.stream().collect(Collectors.summarizingInt(t->t.getId()));
        System.out.println(in);
        System.out.println(in.getAverage()+" "+in.getCount());

        Stream.of(2,4,6,8,9,10,12).takeWhile(n->n%2==0).forEach(System.out::println);  //2 4 6 8
        Stream.of(2,4,6,8,9,10,12).dropWhile(n->n%2==0).forEach(System.out::println); //9 10 12

      }
}
```

Subject: RE: Reg: Day 11 Notes

Classification: **Confidential**

Handson

Exercise 1: Write a lambda expression which accepts x and y numbers and return xy .
Exercise 2: Write a method that uses lambda expression to format a given string, where a space is inserted between each character of string. For ex., if input is “CG”, then expected output is “C G”.
Exercise 3: Write a method that uses lambda expression to accept username and password and return true or false. (Hint: Use any custom values for username and password for authentication)
Exercise 4: Write a class with main method to demonstrate instance creation using method reference. (Hint: Create any simple class with attributes and getters and setters)
Exercise 5: Write a method to calculate factorial of a number. Test this method using method reference feature.

JDK 1.5
1. var args
2. for each stmt
3. Generics
4. Covariant return type
5. static import
6. Autoboxing and Unboxing
7. Annotation
8. Assertion

JDK1.7
1. In switch stmt, we can pass String as an expr
2. try with resources
3. Underscore literal - used for representation

```
    int a=100000;
    int a=1_00_000;  //correct
    int a1=10_000; //correct
    int a2=10_;  //error
```

```
    int a3=_10;  //error
    float f1=10.1_1f; //correct
    float f2=10._11f; //error
    float f3=10_.11f;  //error
    float f4=10.11_f;  //error
```

4. Binary literal

```
    int a=0b11;   //3
    int a1=0B100; //4
```

5. Multi catch statement using |
6. Generics

```
List<String> l=new ArrayList<>();
```

Java 8

1. Lambda expression
   - Used to enable functional programming in Java, until JDK1.7 we cant write anything that exists on its own, first we create a class and access the class with the help of object

```
public class Main {
        public void greet() {
                System.out.println("Hello World");
        }
  public static void main(String[] args) {
    Main m=new Main();
    m.greet();
  }
}
```

Now greet() always print "Hello World", but we need greet() to print different info, until JDK1.7 if we need different implements then we have to go for interface

```
interface Greeting {
        void perform();
}
class HelloWorld implements Greeting {
        public void perform() {
                System.out.println("Helloworld");
        }
}
class Welcome implements Greeting {
        public void perform() {
                System.out.println("Welcome");
        }
}
public class Main {
        /*public void greet() {
                System.out.println("Hello World");
        }*/
        public void greet(Greeting g) {
                g.perform();
        }
  public static void main(String[] args) {
    Main m=new Main();
    Greeting gr=new HelloWorld();  //DMD
    gr.perform(); //Helloworld
    gr=new Welcome();
    gr.perform();  //Welcome
  }
}
```

We want to pass the behaviour (ie) perform() as an argument to greet() and execute the behaviour directly, rather than passing the thing that contains the behaviour (ie) Greeting interface

Lambda expression are just a function that exists on its own and execute it independently, used to enable functional programming

How to write Lambda expr?

1. We take a function and assign to a variable

```
 a=public void perform() {
    System.out.println("Hello");
  }
```

-public, private, protected make sense if we write any method inside class, but in lambda expr, functions are exists on its own so when we write lambda expr there is no need to define access specifiers

```
2. a=void perform() {
        System.out.println("Hello");
   }
```
Whenever we assign a function to a variable, we can access the function using ur variable name,so when we write lambda expr there is no need to define function name

```
3. a=void () {
        //System.out.println("Hello");
         return "hello";
   }
```
By seeing the function itself we can identify the return type of the function,so when we write lambda expr there is no need to define its return type

```
4. a=() {
       System.out.println("Hello");
   }
```

Lambda expr contains parenthesis for input arg, logic and we need to put ->symbol

```
a=() -> {
       System.out.println("Hello");
   }

b=(int a,int b) -> {
          System.out.println(a+b);
   }

5. FunctionType<Void,Void> a =  () -> {
                          System.out.println("Hello");
                         }

FunctionalInterface a= () -> {
                       System.out.println("Hello");
                  }
```

Use a functional interface as a return type for lambda expr, this concept is called Type Inference
- Lambda expr is always used to logic only for methods in functional interface

Rules
1. If ur body of lambda expr is just a single line,then we can ignore curly braces

```
FunctionalInterface a=() ->  System.out.println("hello");
```

2. If ur body of lambda expr is just a single line,then we can ignore the return stmt

```
FunctionalInterface b=(int a) -> {
                             return a*2;
                         }

FunctionalInterface b=(int a) -> a*2;
```

Before JDK1.8, how we can access methods of any interface ? - 2 ways
1. By implementing interface in a class

```
interface Greeting {
          void perform();
}
class Example implements Greeting {
          @Override
          public void perform() {
                    System.out.println("Hello");
          }
}
public class Main {
   public static void main(String[] args) {
     Example e=new Example();
     e.perform();
   }
}
```

2. Using Anonmyous Inner class

```
interface Greeting {
        void perform();
}
public class Main {
   public static void main(String[] args) {
     Greeting g=new Greeting() {
                @Override
                public void perform() {
                        System.out.println("Heloo");
                }
     };
     g.perform();
   }
}
```

Functional Interface
- contains only one abstract method and any number of default and static method
- We will write logic for functionalinterface using lambda expr
- Using @FunctionalInterface annotation - optional - present in java.lang.* pkg
- It is also called as SAM(Single Abstract Method)
- If an interface contains any method of Object class as abstract, then it is also considered as functional interface

```
//Functional interface
interface A {
   void add();
}

//Functional interface
interface A {
   void add();
   default void show() {
   }
   static void show1() {
   }
}

//Not Functional interface
interface A {
   void add();
   int add1();
}

//Functional interface
interface A {
   void add();
   String toString();   //Object class
}

//Not Functional interface
interface A {
   void add();
   boolean equals();
}

//Functional interface
interface A {
   void add();
   boolean equals(Object o);  //Object class
}


//Functional interface
@FunctionalInterface
interface A {
   void add();
}

//Functional interface
interface A {
   void add();
```

```
}
//Not Functional interface
interface B extends A {
  void add1();
}

interface Greeting {
        void perform();
}
public class Main {
   public static void main(String[] args) {
     Greeting gr=()->System.out.println("Using lambda");
     gr.perform();  //Using lambda
   }
}


interface MyInterface {
        int calculateLength(String s);
}
public class Main {
   public static void main(String[] args) {
      MyInterface m1=(e)->e.length();
      System.out.println(m1.calculateLength("hello"));  //5
   }
}

interface MyInterface {
        void add(int a,int b);
}
public class Main {
   public static void main(String[] args) {
      MyInterface m1=(x,y)->System.out.println(x+y);
      m1.add(2, 3);  //5
   }
}
```

2. Method Reference
- also used to access the method of functional interface, instead of writing logic separately using lambda expr, we can refer already existing method, when the signature of interface method is same as static method or instance method or constructor
- Alternative to Lambda expr
- Denoted by ::

3 types
1. Reference to static method
2. Reference to instance method

```
interface MyInterface {
        void show();
}
public class Main {
        /*public static void add() {
                System.out.println("Using reference to static method ");
        }*/
        public void add1() {
                System.out.println("Using reference to instance method ");
        }
   public static void main(String[] args) {
      // MyInterface m1=()->System.out.println("Using Lambda");
      // m1.show(); //Using Lambda

        //MyInterface m2=Main::add;
        //m2.show(); //Using reference to static method

        Main m3=new Main();
        MyInterface m4=m3::add1;
        m4.show();
   }
}
```

3. Reference to a constructor

```
interface MyInterface {
        //Main show();
        Main show(String s);
}
public class Main {
        Main(){
                System.out.println("Default constructor");
        }
        Main(String s){
                System.out.println("Parameterized constructor "+s);
        }
   public static void main(String[] args) {
     //MyInterface m1=Main::new;
     //m1.show();
        MyInterface m2=Main::new;
        m2.show("hello");
   }
}
```

Predefined Functional Interface - present in java.util.function.* pkg

1. Consumer interface
   - void accept(T t)

JDK1.8 - void forEach(Consumer) - used to iterate the elts of List and Set interface individually

2. BiConsumer interface
   - void accept(T t, U u)

JDK1.8 - void forEach(BiConsumer) - used to iterate the elts of Map interface individually

3. Function interface
   - R apply(T t)

4. BiFunction interface
   - R apply(T t,U u)

5. Predicate interface
   - boolean test(T t)

6. BiPredicate interface
   - boolean test(T t,U u)

7. Supplier interface
   - T get()

8. BooleanSupplier interface
   - boolean getAsBoolean()

9. BinaryOperator interface extends BiFunction interface
   - R apply(T t,U u)

10. UnaryOperator interface extends Function interface
   - R apply(T t)

Subject: RE: Reg: Day 10 Notes

Classification: **Confidential**

Handson

1.TreeMap-Player Details

BCCI, for the upcoming IPL season in 2017 decided to give unique cap numbers to every player. Player capNumber is a string. The capNumber and player details are stored in a Treemap.
Create a class named Player with the following private attributes --- name,team and skill. Create a list of objects of Player type.
Cap number is the key and player details is the value. Write a program to display the details of all the players stored in this TreeMap.
Input and Output Format:

Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output:
Enter the number of players
2
Enter the details of the player 1
57
Jaspirit Bumrah
Mumbai Indians
Bowler
Enter the details of the player 2
55
MS Dhoni
Rising Pune Supergiants
All Rounder
Player Details
55--MS Dhoni--Rising Pune Supergiants--All Rounder
57--Jaspirit Bumrah--Mumbai Indians--Bowler

2.TreeMap-Player Details

BCCI, for the upcoming IPL season in 2017 decided to give unique cap numbers to every player. Player capNumber is a string. The capNumber and player details are stored in a Treemap.
Create a class named Player with the following private attributes --- name,team and skill. Create a list of objects of Player type.
Cap number is the key and player details is the value. Write a program to display the details of all the players stored in this TreeMap and details of the players by searching based on cap number.
Input and Output Format:

Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output 1:
Enter the number of players
2
Enter the details of the player 1
57
Jaspirit Bumrah
Mumbai Indians
Bowler
Enter the details of the player 2
55
MS Dhoni
Rising Pune Supergiants
All Rounder
Player Details
55--MS Dhoni--Rising Pune Supergiants--All Rounder
57--Jaspirit Bumrah--Mumbai Indians--Bowler
Enter the cap number of the player to be searched
55
Player Details
MS Dhoni--Rising Pune Supergiants--All Rounder

Sample Input and Output 2:
Enter the number of players
2
Enter the details of the player 1
57
Jaspirit Bumrah
Mumbai Indians
Bowler
Enter the details of the player 2
55
MS Dhoni
Rising Pune Supergiants
All Rounder
Player Details
55--MS Dhoni--Rising Pune Supergiants--All Rounder

57--Jaspirit Bumrah--Mumbai Indians--Bowler
Enter the cap number of the player to be searched
34
Player Details
Player not found

Map interface
- used to store collection of object but in the form of unique key value pair
- Map is unordered

Methods
1. void put(Object k,Object v) - insert single key value pair
2. void putAll(Map m) - store multiple key value pair
3. Object putIfAbsent(Object k,Object v) - if specified key is not associated with the value then it associates the key with given value and returns null
4. boolean containsKey(Object k)
5. boolean containsValue(Object v)
6. Object get(Object k) - if we provide the key, it will return value associated with that key
7. Object getOrDefault(Object k,Object defaultvalue)
8. Object remove(Object k) - if we provide the key, it will remove value associated with that key
9. boolean remove(Object k,Object v)
10. Object replace(Object k,Object v)
11. boolean replace(Object k, Object oldval, Object newval)
12. boolean isEmpty()
13. int size()

We cant apply Iterator interface directly on Map interface, we have to convert Map to Set interface
14. Set entrySet() - convert both key and value to Set interface
15. Set keySet() - convert only keys to Set interface

Map.Entry interface
- used to describe key and value separately
Methods
1. Object getKey()
2. Object getValue()
3. Object setValue(Object k)

SortedMap interface
- used to sort the elements of Map interface
Methods
1. Object firstKey()
2. Object lastKey()
3. SortedMap subMap(Object start,Object end) - start to end-1
4. SortedMap headMap(Object k)
5. SortedMap tailMap(Object k)

3 classes
1. HashMap class
- uses hashing technique to print the value in random order and contains key value pairs
- default capacity is 16
- allow one null key allowed and any number of null values

Syntax: public class HashMap extends AbstractMap implements Map, Cloneable, Serializable

Constructor
1. HashMap()
2. HashMap(int capacity)
3. HashMap(int capacity, float fillratio)
4. HashMap(Map m)

Map<Employee,String> hm=new HashMap<>();

So when we create an object for HashMap, internally it creates a bucket from 0 to 15. Each bucket internally considered as LinkedList, this LinkedList contains one node, the internal structure of node will have key,value,hashcode,next

Employee e1=new Employee(1,"Ram");
Employee e2=new Employee(2,"Sam");
Employee e3=new Employee(3,"Tam");
Employee e4=new Employee(4,"Jam");

hm.put(e1,"HR");

So when we call put(), first it calculate hashcode for the key using hashCode() (ie) hashCode(e1) = 2000
Next it will calculate the index using index=hashcode(key) & (n-1)
=2000 & (16-1) = 2000 & 15 = 6

it stores e1 in index 6

hm.put(e2,"Operation");

So when we call put(), first it calculate hashcode for the key using hashCode() (ie) hashCode(e2) = 1000
Next it will calculate the index using index=hashcode(key) & (n-1)
=1000 & (16-1) = 1000 & 15 = 9
it stores e2 in index 9

hm.put(e3,"Admin");

So when we call put(), first it calculate hashcode for the key using hashCode() (ie) hashCode(e3) = 2000
Next it will calculate the index using index=hashcode(key) & (n-1)
=2000 & (16-1) = 2000 & 15 = 6
it stores e3 in index 6

If same bucket contains multiple node then that is called as Hashing collision, so map will not directly add the entry into the bucket, since both key have same index, map internally calls equals() and check whether the content is same or different, if it is different it will store the entry into same bucket and if it is same it will replace the entry

hm.put(null,"IT");
If we pass key as null, then entry will be directly stored inside 0th bucket

How searching works in LinkedList?

item1:P->item2:P->item3:P->item4:P

get() of HashMap will return the value very faster
when we call hm.get(key), first JVM will find hashcode(key) and from hashcode we can find the bucket index,so jvm will directly goes to the index and find the value, so we use Hashmap because of time complexity
In general, HashMap index can contain multiple nodes in single bucket, ideally have only one node in one bucket. In hash collision, JVM will traverse one by one to fetch the elt, so performance is slow
From Java8, HashMap has been enhanced to reduce the time when using get()

```
Map<Integer,String> hm1=new HashMap<>();
hm1.put(4,"a");
hm1.put(6,"a");
hm1.put(2,"a");
hm1.put(7,"a");
hm1.put(1,"a");
hm1.put(5,"a");
hm1.put(3,"c");
```
Consider all keys have same hashcode so all will have same index in bucket, so performance slow

In Java8, it will convert LinkedList into tree structure, when no of node reaches certain threshold and that threshold is called Treeify threshold

```
Map<Integer,Integer> hm1=new HashMap<>();
hm1.put(4,4);
hm1.put(6,6);
hm1.put(2,2);
hm1.put(7,7);
hm1.put(1,1);
hm1.put(5,5);
hm1.put(3,3);
```

Hence we store integer it takes bigger or smaller, in case of String or object values, so JVM will determine that particular object is greater or smaller using compareTo()

```
public class Main {

	public static void main(String[] args) {
		Map<Integer,String> hm=new HashMap<>();
		System.out.println(hm.size());  //0
		hm.put(10, "apple");
		hm.put(13, "grapes");
		hm.put(15, "banana");
```

```
                hm.put(16, "Guava");
                hm.put(10, "mango"); //no error, the value will be overridden
                System.out.println(hm.size());  //4
                System.out.println(hm); //print in curly braces, in random order

                System.out.println(hm.putIfAbsent(20, "Melon")); //null
                System.out.println(hm.putIfAbsent(11, "Berry")); //null
                System.out.println(hm.putIfAbsent(10, "Lime")); //mango
                System.out.println(hm.size());  //6
                System.out.println(hm.get(16)); //Guava
                System.out.println(hm.get(17));  //null
                System.out.println(hm.getOrDefault(17, "Pineapple")); //Pineapple
                System.out.println(hm.size()); //6
                hm.remove(10);
                System.out.println(hm.get(10)); //null
                System.out.println(hm.remove(11, "Berry")); //true
                System.out.println(hm.get(11)); //null
                System.out.println(hm.remove(13,"Berry")); //false
                System.out.println(hm.get(13)); //grapes

                System.out.println(hm.replace(16, "Guava", "Orange")); //true
                System.out.println(hm.get(16)); //Orange
                System.out.println(hm.replace(20, "Greenapple")); //Melon
                System.out.println(hm.get(20)); //Greenapple
                System.out.println(hm);

                Set set=hm.entrySet();  //convert both key and value to Set intf
                Iterator itr=set.iterator();
                while(itr.hasNext()) {
                        Map.Entry me=(Map.Entry)itr.next();  //one key value pair
                        System.out.println(me.getKey()+" "+me.getValue());
                }

                set=hm.keySet();  //convert only key to Set intf
                Iterator<Integer> itr1=set.iterator();
            while(itr1.hasNext()) {
                Integer key=itr1.next();  //one key
                System.out.println(key+" "+hm.get(key));
            }
        }

}
```

2. LinkedHashMap class
   - maintain the insertion order

Syntax: public class LinkedHashMap extends HashMap

Constructor
1. LinkedHashMap()
2. LinkedHashMap(int capacity)
3. LinkedHashMap(int capacity, float fillratio)
4. LinkedHashMap(Map m)

3. TreeMap class
   - used to sort the elemts only based on key
   - u cant add null key, we can have null values

Syntax: public class TreeMap extends AbstractMap implements NavigableMap, Cloneable, Serializable

4. HashTable class
   - It is a legacy class
   - similar to HashMap but it is synchronized or threadsafe

Syntax: public class Hashtable extends Dicitionary implements Map, Cloneable, Serializable

Constructor
1. Hashtable()
2. Hashtable(int capacity)
3. Hashtable(int capacity, float fillratio)
4. Hashtable(Map m)

5. Properties class
- contains key value pairs but both key and value should be in the form of String, print in random order

Syntax: public class Properties extends Hashtable

Constructor
1.Properties()
2.Properties(String default)

```
public class Main {

	public static void main(String[] args) {
		LinkedHashMap<Integer,String> hm=new LinkedHashMap<>();
		System.out.println(hm.size());  //0
		hm.put(10, "apple");
		hm.put(13, "grapes");
		hm.put(15, "banana");
		hm.put(16, "Guava");
		hm.put(10, "mango"); //no error, the value will be overridden
		System.out.println(hm);

		TreeMap<String,Integer> tm=new TreeMap<>();
		tm.put("lion",10);
		tm.put("cat",30);
		tm.put("ape",20);
		tm.put("bear",40);
		System.out.println(tm);

		Hashtable<Integer,Double> ht=new Hashtable<>();
		System.out.println(ht.size());
		ht.put(3, 2.45);
		ht.put(1, 5.67);
		ht.put(6, 4.23);
		ht.put(7, 1.23);
		System.out.println(ht);

		Enumeration<Integer> e=ht.keys();
		while(e.hasMoreElements()) {
			Integer key=e.nextElement();
			System.out.println(key+" "+ht.get(key));
		}

		Properties p=new Properties();
		p.put("cat", "milk");
		p.put("dog", "meat");
		p.put("cow", "grass");
		p.put("fish", "water");
		System.out.println(p);

		Set set=p.keySet();  //only key
		Iterator<String> itr=set.iterator();
		while(itr.hasNext()) {
			String key=itr.next();
			System.out.println(key+" "+p.getProperty(key));
		}
	}

}
```

Subject: RE: Reg: Day 9 Notes

Classification: **Confidential**

Handson

1. Write a Java program to read all the player information from the user and display the player name and their cap number sorted based on their cap number (descending order). The contact details consist of player name, skill and cap number. Use Collections.sort() method for sorting.

Create a main class "Main.java"
Create Player class with below private attributes

playerName - String

skill - String

capNumber - Long

Add appropriate getter and setter methods for Player class
Include a constructor for Player class with the arguments player name, skill and cap number
Implement Comparable interface and implement the method compareTo() to perform sorting based on cap number

Input and Output Format:
First input corresponds to the number of players and followed by each player's information.
Refer sample input and output for formatting specifications.

[All text in bold corresponds to input and the rest corresponds to output]
Sample Input/Output :
Enter number of players:
2
Enter player 1 detail
Enter Name
Suresh Raina
Enter Skill
Batsman
Enter Cap Number
265
Enter player 2 detail
Enter Name
Virat Kohli
Enter Skill
Batsman
Enter Cap Number
268
Player list after sorting by cap number in descending order
Virat Kohli-268
Suresh Raina-265

2.Comparator - no of matches player

Write a Java program to get the team name and number of matches played by the team from the user and display a report with team name and number of matches sorted based on the number of matches in ascending order. Use Collection.sort() method to perform the sorting in your main class. Send the Comparator object as second argument to the sort method to use this comparator for sorting.

Create a main class " Main.java"
Create a class named Team with the following private member variables / attributes,
name - String
numberOfMatches - Long
Include a constructor accepting Team name and number of matches as arguments
Add appropriate getter and setter methods for Team class

Create TeamComparator implementing Comparator interface
Implement compare method to compare two team objects based on their number of matches played.

Input and Output Format:
First input corresponds to the number of teams and followed by each team information.
Refer sample input and output for formatting specifications.

[All text in bold corresponds to input and the rest corresponds to output]
Sample Input/Output :
Enter number of teams:
3
Enter team 1 detail
Enter Name
Chennai super Kings
Enter number of matches
132

```
Enter team 2 detail
Enter Name
Royal Challengers Bangalore
Enter number of matches
139
Enter team 3 detail
Enter Name
Delhi Daredevils
Enter number of matches
131
Team list after sort by number of matches
131
132
139
```

Collections class
- contain static methods/algorithm that support util package

Methods
1. static int binarySearch(List l,int val) - search any elet in list
2. static void copy(List dest,List src)
3. static List nCopies(int val,Object o) - create an immutable list with list of values
4. static boolean disjoint(List l1,List l2) - return true if there is no common elt
5. static List emptyList() - create an immutable list
6. static Set emptySet() - create an immutable set
7. static Map emptyMap() - create an immutable map
8. static void fill(List l, Object o) - fill the list with particular value
9. static int frequency(List l, Object o) - return how many times an elt occur in the list
10. static boolean replaceAll(List l,Object oldval, Object newval)
11. static Object max(List l) - find max elt
12. static Object min(List l) - find min elt
13. static void reverse(List l)
14. static void shuffle(List l) - randomly shuffle elt
15. static List singletonList(Object o) - create list with single elt
16. static Set singleton(Object o)
17. static Map singletonMap(Object k, Object v)
18. static void sort(List l)
19. static void sort(List l,Comparator c)
20. static void swap(List l, int oldindex, int newindex)
21. static List synchronizedList(List l)
22. static Set synchronizedSet(Set s)
23. static Map synchronizedMap(Map m)
24. static List unmodifiableList(List l)
25. static Set unmodifiableSet(Set s)
26. static Map unmodifiableMap(Map m)

```
public class Main {
        public static void main(String...args) {
    List<Integer> l1=new ArrayList<>();
    l1.add(3);
    l1.add(5);
    l1.add(6);
    l1.add(7);
    l1.add(9);
    System.out.println(Collections.binarySearch(l1, 6)); //2

    List<String> l2=new ArrayList<>();
    l2.add("one");
    l2.add("two");
    l2.add("three");
    List<String> l3=new ArrayList<>();
    l3.add("four");
    l3.add("five");
    l3.add("six");
    //l3.add("seven");

    System.out.println(Collections.disjoint(l2, l3));  //true
    Collections.copy(l3, l2);
    System.out.println(l2);  //[one,two,three]
```

```
        System.out.println(l3);  //[one,two,three]
        System.out.println(Collections.disjoint(l2, l3)); //false

        List l4=Collections.nCopies(3, "Hello");
        System.out.println(l4); //[Hello,Hello,Hello]
        //l4.add("Hi");
        //System.out.println(l4); //UnsupportedOperationException

        List l5=Collections.emptyList();
        System.out.println(l5);  //[]
        //l5.add("Hi");
       // System.out.println(l5);  //exception

        System.out.println(Collections.frequency(l4, "Hello"));  //3
        Collections.fill(l3,"Hi");
        System.out.println(l3); //[Hi,Hi,Hi]
        Collections.replaceAll(l3, "Hi", "He");
        System.out.println(l3);  //[He,He,He]

        List<String> l6=new ArrayList<>();
        l6.add("Hii");
        l6.add("Hi");
        l6.add("Hiiii");
        l6.add("Hi Good monring");
        Collections.replaceAll(l6, "Hi", "He");
        System.out.println(l6);  //[Hii,He,Hiiii,Hi Good morning]

        List<Integer> l7=new ArrayList<>();
        l7.add(8);
        l7.add(-20);
        l7.add(-8);
        l7.add(20);
        System.out.println(l7);  //[8,-20,-8,20]

       Collections.reverse(l7);
       System.out.println(l7);  //[20,-8,-20,8]
       System.out.println(Collections.max(l7));  //20
       System.out.println(Collections.min(l7)); //-20

       Collections.shuffle(l7);
       System.out.println(l7);  //random order
       Collections.sort(l7); //ascending order
       System.out.println(l7); //[-20,-8,8,20]

       Collections.sort(l7,Comparator.reverseOrder()); //descending order
       System.out.println(l7); //[20,8,-8,-20]

       Collections.swap(l7, 0, 3);
       System.out.println(l7); //[20,-8,8,-20]

       l7=Collections.singletonList(23);
       System.out.println(l7); //[23]
       //l7.add(24);
      // System.out.println(l7);  //exception

       List l8=Collections.synchronizedList(l6);
       System.out.println(l8);  //threadsafe

       List l9=Collections.unmodifiableList(l2);
       System.out.println(l9);  //[one,two,three] - immutable
       l9.add("Hello");
       System.out.println(l9); //exception
            }
}
```

Arrays class
- used to perform operation on any datatype array

Methods
1. static List asList(int[] a) - convert int array to List
2. static void fill(int[] a, int val) - fill all elts of array with val

3. static void fill(int[] a,int start,int end,int val) - fill elt with val from start to end-1
4. static void sort(int[] a) - sort all elts of array
5. static void sort(int[] a, int start,int end) - sort elts from start to end
6. static boolean equals(int[] a1,int[] a2)
7. static int binarySearch(int[] a,int val)

```
public class Main {
	public static void main(String...args) {
    Integer a[]=new Integer[10];
    for(int i=0;i<10;i++) {
        a[i]=i*-3;
    }
    for(int a1:a)
        System.out.println(a1);

    List<Integer> l=Arrays.asList(a);
    System.out.println(l); // [0,-3,-6,-9,-12,-15,-18,-21,-24,-27]

    Arrays.sort(a); //[-27,-24,-21,-18,-15,-12,-9,-6,-3,0]
    System.out.println(Arrays.binarySearch(a, -15)); //4

    //If value is not present it returns negative value based on -(index)-1
    //[-27,-24,-21,-18,-15,-12,-10,-9,-6,-3,0] = -(6)-1
    System.out.println(Arrays.binarySearch(a, -10)); //-7
    //[-27,-24,-22,-21,-18,-15,-12,-10,-9,-6,-3,0] = -(2)-1
    System.out.println(Arrays.binarySearch(a, -22));  //-3

    Arrays.sort(a); //[-27,-24,-21,-18,-15,-12,-9,-6,-3,0]
    Arrays.fill(a,2,5,-1); //start to end-1 by -1
     //[-27,-24,-1,-1,-1,-12,-9,-6,-3,0]
    Arrays.sort(a,2,5); //[-27,-24,-12,-1,-1,-1,-9,-6,-3,0]
    Arrays.sort(a); //[-27,-24,-12,-9,-6,-3,-1,-1,-1,0]
        }
}
```

- If we want to sort the elts of List interface, we have to use Collections.sort(List l)
- If we want to sort the elts of Set interface, we have to use TreeSet class
- If it is numbers then it will sorted in ascending order and String means it will sorted in alphabetical order

Comparable and Comparator interface
- If ur List or Set interface contains collection of user defined object like Employee, Student, Player etc, now we want to sort based on their properties rather than sorting its reference

List<Employee> l=new ArrayList<>();
empid,ename,salary,age - properties

Comparable interface	Comparator interface
1. provides single sorting sequence (ie) we can do sorting based on only property either by empid or ename or salary or age	1. provides multiple sorting sequence (ie) we can do sorting based on empid and name and salary age
2. present in java.lang.*	2. present in java.util.*
3. int compareTo(Object o)	3. int compare(Object o1,Object o2)
4. We do sorting using Collections.sort(List l) for List intf and TreeSet class for Set intf	4. We do sorting using Collections.sort(List l,Comparator c) for List intf and TreeSet(Comparator c) for Set intf

```
public class Student implements Comparable<Student> {
  private Integer stuId;
  private String name;
  private Integer age;
public Integer getStuId() {
	return stuId;
}
public void setStuId(Integer stuId) {
	this.stuId = stuId;
```

```
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}
	public Student(Integer stuId, String name, Integer age) {
		super();
		this.stuId = stuId;
		this.name = name;
		this.age = age;
	}
	public Student() {
		super();
		// TODO Auto-generated constructor stub
	}
	@Override
	public int compareTo(Student o) {
		// TODO Auto-generated method stub
		return name.compareTo(o.name);
	}
	/*@Override
	public int compareTo(Student o) {
		if(age==o.age)
		  return 0;
		else if(age>o.age)
			return 1;
		else
			return -1;
	}*/
	@Override
	public String toString() {
		return "Student [stuId=" + stuId + ", name=" + name + ", age=" + age + "]";
	}

	}

	public class Employee {
		Integer eId;
		  String name;
		   Integer age;
		public Integer getEId() {
			return eId;
		}
		public void setEId(Integer eId) {
			this.eId = eId;
		}

		public String getName() {
			return name;
		}
		public void setName(String name) {
			this.name = name;
		}
		public Integer getAge() {
			return age;
		}
		public void setAge(Integer age) {
			this.age = age;
		}
		public Employee(Integer eId, String name, Integer age) {
			super();
			this.eId = eId;
			this.name = name;
```

```
		this.age = age;
	}
	public Employee() {
		super();
		// TODO Auto-generated constructor stub
	}
	@Override
	public String toString() {
		return "Employee [eId=" + eId + ", name=" + name + ", age=" + age + "]";
	}



}


public class AgeComparator implements Comparator<Employee> {

	@Override
	public int compare(Employee o1, Employee o2) {
		if(o1.age==o2.age)
		  return 0;
		else if(o1.age>o2.age)
			return 1;
		else
			return -1;
	}

}


public class NameComparator implements Comparator<Employee>{

	@Override
	public int compare(Employee o1, Employee o2) {
		return o1.name.compareTo(o2.name);
	}

}

public class Main {
	public static void main(String...args) {
      List<Integer> l1=new ArrayList<>();
      l1.add(4);
      l1.add(2);
      l1.add(5);
      l1.add(1);
      l1.add(6);
      System.out.println(l1);  //[4,2,5,1,6]
      Collections.sort(l1);
      System.out.println(l1);  //[1,2,4,5,6]
      Collections.sort(l1,Comparator.reverseOrder());
      System.out.println(l1);  //[6,5,4,2,1]

      TreeSet<String> ts=new TreeSet<>();
      ts.add("H");
      ts.add("B");
      ts.add("E");
      ts.add("A");
      System.out.println(ts); //[A,B,E,H]
      TreeSet ts1=(TreeSet) ts.descendingSet();
      System.out.println(ts1); //[H,E,B,A]

      List<Student> l2=new ArrayList<>();
     l2.add(new Student(103,"Ram",24));
      l2.add(new Student(101,"Sam",20));
      l2.add(new Student(104,"Raj",21));
      l2.add(new Student(100,"Tam",23));
      l2.add(new Student(102,"Jam",19));

      Collections.sort(l2);  //internally calls compareTo
```

```
        for(Student s1:l2)
            System.out.println(s1);

        System.out.println();
        TreeSet<Student> ts2=new TreeSet<>();
        ts2.add(new Student(103,"Ram",24));
        ts2.add(new Student(101,"Sam",20));
        ts2.add(new Student(104,"Raj",21));
        ts2.add(new Student(100,"Tam",23));
        ts2.add(new Student(102,"Jam",19));
        for(Student s1:ts2)
            System.out.println(s1);

        List<Employee> l3=new ArrayList<>();
        l3.add(new Employee(103,"Ram",24));
        l3.add(new Employee(101,"Sam",20));
        l3.add(new Employee(104,"Raj",21));
        l3.add(new Employee(100,"Tam",23));
        l3.add(new Employee(102,"Jam",19));

        Collections.sort(l3,new AgeComparator());
        for(Employee e1:l3)
            System.out.println(e1.getAge());

        System.out.println();
        Collections.sort(l3,new NameComparator());
        for(Employee e1:l3)
            System.out.println(e1.getName());

        System.out.println();
        TreeSet<Employee> ts3=new TreeSet<>(new AgeComparator());
        ts3.add(new Employee(103,"Ram",24));
        ts3.add(new Employee(101,"Sam",20));
        ts3.add(new Employee(104,"Raj",21));
        ts3.add(new Employee(100,"Tam",23));
        ts3.add(new Employee(102,"Jam",19));
        for(Employee e1:ts3)
            System.out.println(e1);
            }
}
```

Map interface
- used to store collection of object but in the form of unique key value pair
- Map is unordered

Methods
1. void put(Object k,Object v) - insert single key value pair
2. void putAll(Map m) - store multiple key value pair
3. Object putIfAbsent(Object k,Object v) - if specified key is not associated with the value then it associates the key with given value and returns null
4. boolean containsKey(Object k)
5. boolean containsValue(Object v)
6. Object get(Object k) - if we provide the key, it will return value associated with that key
7. Object getOrDefault(Object k,Object defaultvalue)
8. Object remove(Object k) - if we provide the key, it will remove value associated with that key
9. boolean remove(Object k,Object v)
10. Object replace(Object k,Object v)
11. boolean replace(Object k, Object oldval, Object newval)
12. boolean isEmpty()
13. int size()
14. Set entrySet()
15. Set keySet()

Map.Entry interface
- used to describe key and value separately

Methods
1. Object getKey()
2. Object getValue()
3. Object setValue(Object k)

SortedMap interface
- used to sort the elements of Map interface

Methods
1. Object firstKey()
2. Object lastKey()
3. SortedMap subMap(Object start,Object end) - start to end-1
4. SortedMap headMap(Object k)
5. SortedMap tailMap(Object k)

Subject: RE: Reg: Day 8 Notes

Classification: **Confidential**

Handson

1. The Chennai Super Kings were the most successful team in the IPL with a win percentage of 60.68 and had won the title twice in succession (2010 and 2011).

The CSKs have played “n” matches so far in IPLs. Given the number of matches “n” that CSK has played and their team score in the matches as a list, write a program to find the total runs and the average runs scored by the team in all “n” matches.

Input Format:
First line of the input is an integer “n” that corresponds to the number of matches played by CSK.
Next “n” lines contains an integer in each line, that corresponds to the runs scored by CSK in each of the “n” matches.

Output Format:
Output should print in the first line the integer that gives the total runs scored by CSK in the matches.
In the second line, print a float value that gives the average runs.

Sample Input :
5
200
210
180
176
192

Sample Output :
958
191.6

2. A measure of a batsman's greatness is his ability to score runs on foreign conditions. It is quite obvious in Cricket that most batsmen have been excellent at home grounds but flops overseas. Consequently not many teams have aggregates balanced in terms of home and away performances.

Sunil now wanted to analyze the performance of IPL teams based on the runs scored in home as well away matches. Given are the team name, number of matches played by the team in home ground “n” and away grounds “m” respectively, runs scored by the team in each of the matches both home and away respectively. Write a program to store the runs scored by the team in both home ground and in other grounds in a list and help Sunil to display the score (in both home and away grounds) of the team that is greater than 300.

Input Format:

First line of the input contains a string that gives the name of the IPL team.

Second line of input contains the integer “n” that corresponds to the number of matches played by the team in home grounds.

Next “n” lines contains the runs scored by the team in each of the matches in home grounds.

Next line that follows contains the integer “m” that corresponds to the number of matches played by the team in away grounds.

Next “m” lines contains the runs scored by the team in each of the matches in away grounds.

Output Format:

Output should display the runs scored by the team in both home ground and in other grounds as a list, line after line.

In the lines to follow, the output should display the score (in both home and away grounds) of the team that is greater than 300, line after line.

Sample Input and output :

```
Enter the team name
Chennai Super Kings
Enter the number of matches played in home ground
2
Enter the runs scored
290
320
Enter the number of matches played in other ground
3
Enter the runs scored
399
180
150
Runs scored by Chennai Super Kings
290
320
399
180
150
Run scored by Chennai Super Kings more than 300
320
399
```

java.util.*
- Utility framework or collection framework
- used to store collection of objects - any objects (ie) Employee, Student, String, Player, Venue

1. Collection interface - core interface in order to store collection of object
2. Collections class - provided with static methods or static algorithm that supports util package

Collection interface
- core interface in order to store collection of object

Methods
1. boolean add(Object o) - add single object
2. boolean addAll(Collection c) - add multiple object
3. boolean remove(Object o) - remove single object
4. boolean removeAll(Collection c) - remove multiple object
5. boolean contains(Object o) - check single object present in Collection or not
6. boolean containsAll(Collection c) - check multiple object present in Collection or not
7. boolean retainAll(Collection c)- remove from the target collection all the elemts that are not contained in the specified collection
8. int size() - return number of objects present in the collection
9. boolean isEmpty() - to check collection is empty or not
10. Object[] toArray() - return a array containing all elts in collection
11. String toString()
12. boolean equals(Object o)
13. Iterator iterator()
14. ListIterator listIterator()

List interface
- Ordered and duplicate elements allowed

Methods
1. void add(int index,Object o) - add single object at particular index position
2. boolean addAll(int index,Collection c) - add multiple object at particular index position
3. Object get(int index) - return single object present at particular index position
4. int indexOf(Object o) - return position of first occurence of object in given collection
5. int lastIndexOf(Object o) - return position of last occurence of object in given collection
6. Object remove(int index) - remove single object present at particular index position
7. Object set(int index,Object val) - replace an object at particular index
8. List subList(int start,int end) - return part of list from start to end-1

Set interface
- unordered and no duplicate elets allowed

SortedSet interface
- sort the elts of Set interface

Methods
1. Object first() - return first elt
2. Object last() - return last elt
3. SortedSet subSet(Object from, Object end) - return part of set from start to end-1
4. SortedSet headSet(Object elt) - return all elts present before specified elt
   (1 2 3 4 5).headSet(3) //1 2

5. SortedSet tailSet(Object elt) - return all elts present after specified elt
   (1 2 3 4 5).tailSet(3) //4 5

List interface
- Ordered and duplicate elements allowed

int a[]=new int[3]; - Static array

1. ArrayList class
- It is similar to array but it is called as dynamic array (ie) we can increase or decrease its size at runtime

Syntax: public class ArrayList extends AbstractList implements List, Cloneable, Serializable, RandomAccess

- used for faster retrieval of data and slower in insertion and deletion
- default capacity is 10

Constructor
1. ArrayList()
2. ArrayList(int capacity)
3. ArrayList(Collection c)

```
public class Main {
        public static void main(String...args) {
           ArrayList l1=new ArrayList();
           System.out.println(l1.size());  //0
           l1.add("A");
           l1.add("B");
           l1.add(10);   //Autoboxing
           System.out.println(l1.size());  //3
           //wheneever we print object of any collection inside []
           System.out.println(l1); //[A,B,10]

           ArrayList l2=new ArrayList();
           System.out.println(l2.size()); //0
           l2.addAll(l1);
           System.out.println(l2); //[A,B,10]

           System.out.println(l1.contains(10));  //true
           System.out.println(l1.containsAll(l2)); //true
           System.out.println(l1.indexOf(10)); //2

           List l3=new ArrayList();  //DMD
           l3.add("A");
           l3.add("B");
           l3.add(1);
           l3.add(2);
           l3.add(3);
           System.out.println(l3);  //[A,B,1,2,3]

           List l4=new ArrayList();
           l4.add("A");
           l4.add("B");
           l4.add(1);
           System.out.println(l4); //[A,B,1]

           l3.retainAll(l4);
           System.out.println(l3); //[A,B,1]
        }
}
```

Generics
- Available from JDK1.5 onwards
- used to specify what type of object stored in the collection, avoid type casting
- denoted by <>

```
List<String> l1=new ArrayList<String>();  //JDK1.5
List<Integer> l2=new ArrayList<>(); //From JDK1.7

public class Main {
	public static void main(String...args) {
	  List<String> l1=new ArrayList<>();
	  System.out.println(l1.size());  //0
	  l1.add("C");
	  l1.add("F");
	  l1.add(1,"O");
	  l1.add("T");
	  l1.add("R");
	  //C,O,F,T,R
	  System.out.println(l1); //[C,O,F,T,R]
	  System.out.println(l1.get(2));  //F
	  System.out.println(l1.contains("S"));  //false
	  List<String> l2=new ArrayList<>();
	  l2.add("C");
	  l2.add("F");
	  System.out.println(l1.containsAll(l2)); //true
	  System.out.println(l1.indexOf("T")); //3
	  l1.remove("T");  //[C,O,F,R]
	  l1.remove(1);
	  System.out.println(l1); //[C,F,R]

	}
}
```

2. LinkedList class
   - It is also called dynamic array
   - used for faster insertion and deletion, slower in selection

Syntax: public class LinkedList extends AbstractSequentialList implements List, Cloneable, Serailizable, Deque

Deque - Double Ended Queue - do insertion and deletion at both ends

Constructors
1. LinkedList()
2. LinkedList(Collection c)

Methods
1. void addFirst()
2. void addLast()
3. Object getFirst()
4. Object getLast()
5. Object removeFirst()
6. Object removeLast()
7. void push(Object o)
8. Object pop()
9. boolean offerFirst(Object o) - insert specified elt front of the list
10. boolean offerLast(Object o) - insert specified elt last of the list
11. Object peekFirst() - retrieves first elt of the list but does not remove
12. Object peekLast() - retrieves last elt of the list but does not remove
13. Object pollFirst() - retrieves first elt of the list and remove
14. Object pollLast() - retrieves last elt of the list and remove

```
public class Main {
	public static void main(String...args) {
	  LinkedList<Integer> l1=new LinkedList<>();
	  l1.add(3);
	  l1.add(5);
	  l1.add(2);
	  l1.add(1,7);
	  l1.addFirst(8);
	  l1.addLast(9);
	  System.out.println(l1.size()); //6
	  System.out.println(l1); //[8,3,7,5,2,9]
	  System.out.println(l1.get(5)); //9
	  System.out.println(l1.getFirst()); //8
	  System.out.println(l1.getLast()); //9
```

```
        l1.remove(1); //8,7,5,2,9
        l1.removeFirst(); //7,5,2,9
        l1.removeLast();
        System.out.println(l1); //[7,5,2]
        System.out.println(l1.peekFirst()); //7
        System.out.println(l1.pollLast()); //2
        System.out.println(l1); //[7,5]
      }
}
```

3. Vector class
   - It is a legacy(older) class
   - It is also called dynamic array, but it is synchronized or threadsafe
   - Default capacity is 10

Syntax: public class Vector extends AbstractList implements List,Cloneable, Serailizable

Constructor
1. Vector()
2. Vector(int capacity)
3. Vector(int capacity,int increment)
4. Vector(Collection c)

```
public class Main {
      public static void main(String...args) {
        Vector<Integer> v=new Vector<>(3,2);
        System.out.println(v.size()); //0
        System.out.println(v.capacity()); //3
        v.add(1);
        v.add(2);
        v.add(3);
        v.add(4);
        System.out.println(v.size()); //4
        System.out.println(v.capacity()); //5
        v.add(5);
        v.add(6); //5+2
        System.out.println(v.size()); //6
        System.out.println(v.capacity()); //7
        System.out.println(v); //[1,2,3,4,5,6]
      }
}
```

Set interface
- unordered and no duplicates allowed

1. HashSet class
   - does not maintain any insertion order and the elts are inserted in the basis of their hashcode value
   - default capacity is 16

Syntax: public class HashSet extends AbstractSet implements Set, Cloneable, Serializable

Constructor
1. HashSet()
2. HashSet(int capacity)
3. HashSet(int capacity,float fillratio)
   fillratio ranges from 0.0 to 1.0 - default is 0.75
4. HashSet(Collection c)

```
Set<Integer> hs=new HashSet<>();      Map<Integer,Object> hm=new HashMap<>();
sop(hs.size()); //0
hs.add(2);                    hm.put(2,PRESENT);
hs.add(3);                    hm.put(3,PRESENT);
hs.add(2);                    hm.put(2,PRESENT);
sop(hs.size());  //2
sop(hs); //[2,3]
```

When we create HashSet, it internally creates HashMap(unique key value pair), with key as what elt we are inserting into the hashset and value as dummy object called PRESENT

If hm.put(key,value) returns null then the statement hm.put(2,PRESENT)==null will return true then the element will be added into hashset

If hm.put(key,value) returns old value of key then the statement hm.put(2,PRESENT)==null will return false then the element will not be added into hashset

```
Set<Integer> hs=new HashSet<>(3);
hs.add(1);
hs.add(2);
hs.add(3);
hs.add(4);  3*0.75=2.25+3=5.25
hs.add(5);
hs.add(6); 5.25*0.75=3+5.25=8.25
```

2. LinkedHashSet class
   - used to maintain the insertion order of the set

Syntax: public class LinkedHashSet extends HashSet implements Set,Cloneable, Serializable

Constructor
1. LinkedHashSet()
2. LinkedHashSet(int capacity)
3. LinkedHashSet(int capacity,float fillratio)
   fillratio ranges from 0.0 to 1.0 - default is 0.75
4. LinkedHashSet(Collection c)

3. TreeSet class
   - Print the elements of Set interface in Sorted format(ie)numbers in ascending, string in alphabetical order

Syntax: public class TreeSet extends AbstractSet implements NavigableSet, Cloneable, Serializable

Constructor
1. TreeSet()
2. TreeSet(Comparator c)
3. TreeSet(Collection s)
4. TreeSet(SortedSet s)

```
public class Main {
        public static void main(String...args) {
      Set<Integer> hs=new HashSet<>();
       hs.add(20);
       hs.add(3);
       hs.add(11);
       hs.add(4);
       hs.add(1);
       System.out.println(hs); //random order

       LinkedHashSet<Integer> ls=new LinkedHashSet<>();
       ls.add(20);
       ls.add(3);
       ls.add(11);
       ls.add(4);
       ls.add(1);
       System.out.println(hs); //[20,3,11,4,1]

       TreeSet<Integer> ts=new TreeSet<>();
       ts.add(20);
       ts.add(3);
       ts.add(11);
       ts.add(4);
       ts.add(1);
       System.out.println(ts); //[1,3,4,11,20]
       TreeSet ts1=(TreeSet)ts.descendingSet();
       System.out.println(ts1); //[20,11,4,3,1]
         }
}
```

To print individual elements of collection - 5 ways
1. Normal for loop
2. for each stmt
3. Using Iterator interface
   - used to print individual elements of collection only in forward direction
   - Using Iterator iterator(), we can create object for Iterator
   - Methods

1. boolean hasNext() - check whether it contains next next elt
2. Object next() - retrieve each elt
3. void remove() - at time of accessing even we can remove the elt

4. Using ListIterator interface
- used to print individual elements of collection both in forward and backward direction
- Using ListIterator listIterator(), we can create object for ListIterator
- Methods
1. boolean hasNext() - check whether it contains next next elt
2. Object next() - retrieve each elt
3. void remove() - at time of accessing even we can remove the elt
4. boolean hasPrevious()
5. Object previous()
6. void set(Object o)
7. void add(Object o)

5. Using Enumeration interface
- It is legacy interface
- used to print individual elements of collection only in forward direction
- Using Enumeration elements(), we can create object for Enumeration
- Methods
1. boolean hasMoreElements() - check whether it contains next next elt
2. Object nextElement() - retrieve each elt

```
public class Main {
    public static void main(String...args) {
    List<Integer> l1=new ArrayList<>();
    l1.add(30);
    l1.add(20);
    l1.add(10);
    l1.add(24);
    System.out.println(l1);  //[30,20,10,24]

    //1. Normal FOR loop
    for(int i=0;i<l1.size();i++) {
        System.out.println(l1.get(i));
    }

    System.out.println("Using For each stmt");
    for(Integer i:l1)
        System.out.println(i);

    System.out.println("Using Iterator");
    Iterator<Integer> i=l1.iterator();
    while(i.hasNext()) {
        // Integer i1=(Integer)i.next();
          Integer i1=i.next();
         System.out.println(i1);
    }

    List<Student> l2=new ArrayList<>();
    l2.add(new Student(1,"Ram",23));
    l2.add(new Student(2,"Sam",21));
    l2.add(new Student(3,"Tam",20));
    l2.add(new Student(4,"Jam",24));

    ListIterator<Student> li=l2.listIterator();
    while(li.hasNext()) {
        Student s1=li.next();
        System.out.println(s1.getName());
    }
    System.out.println();
    while(li.hasPrevious()) {
        Student s1=li.previous();
        System.out.println(s1.getName());
    }

    List<String> l3=new ArrayList<>();
    l3.add("Ram");
    l3.add("Sam");
    l3.add("Tam");
    ListIterator<String> lt=l3.listIterator();
```

```
        while(lt.hasNext()) {
            String s1=lt.next();
            lt.set(s1+"mmm");
        }
        while(lt.hasPrevious()) {
            String s1=lt.previous();
            System.out.println(s1);
        }

        Vector<Integer> v=new Vector<>();
          v.add(1);
          v.add(2);
          v.add(3);
          System.out.println(v); //[1,2,3]
          Enumeration e=v.elements();
          while(e.hasMoreElements()) {
                    System.out.println(e.nextElement());
          }
          }
}
```

Subject: RE: Reg: Day 7 Notes

Classification: **Confidential**

Handson

1. Exception 2(ArrayIndexOutOfBoundsException And NegativeArraySizeException)

Write a program to get the number of overs and the runs scored in each over. Get the over number from the user and display number of runs scored in that over. Let

- number of overs be the array size
- over number be the index of the array+1
- runs be the array elements

This program may generate
1. NegativeArraySize Exception when the number of overs is negative
2. ArrayIndexOutOfRange Exception when the over number that is searched is beyond the specified over numbers.

Use exception handling mechanisms to handle these exceptions.Use a single catch block. In the catch block, print the class name of the exception thrown.
Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input/Output 1:
Enter the number of overs
3
Enter the number of runs for each over
8
15
12
Enter the over number
2
Runs scored in this over : 15

Sample Input/Output 2:
Enter the number of overs
3
Enter the number of runs for each over
8
15
12
Enter the over number
4
java.lang.ArrayIndexOutOfBoundsException

Sample Input/Output 3:
Enter the number of overs
-1
Enter the number of runs for each over
java.lang.NegativeArraySizeException

2.Custom Exceptions [Age]

Write a program to get the name and age of the player from the user and display it.
player name is a string
player age is an integer value
Note : The player is eligible to participate in IPL when their age is 19 and above

This program may generate
1. InvalidAgeRange Custom Exception when the player's age is below 19
Use exception handling mechanisms to handle these exceptions

Create a class called CustomException which extends Exception and it includes constructor to initialize the message.

Use appropriate exception handling mechanisms to handle these exceptions

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input/Output 1:
Enter the player name
Albie Morkel
Enter the player age
35
Player name : Albie Morkel
Player age : 35

Sample Input/Output 2:
Enter the player name
Ishan Kishan
Enter the player age
16
CustomException: InvalidAgeRangeException

3. TeamNameNotFound Exception

Write a program to get the two team names i.e expected Runner and Winner team of IPL season 4 and display it.
Team name is a string

Note : The team name given below are only eligible to take part in IPL season 4
Chennai Super Kings
Deccan Chargers
Delhi Daredevils
Kings XI Punjab
Kolkata Knight Riders
Mumbai Indians
Rajasthan Royals
Royal Challengers Bangalore
This program may generate TeamNameNotFound Custom Exception when the expected team entered is not present in the above eligible teams list for IPL season 4.
Use exception handling mechanisms to handle these exceptions

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.
Sample Input and Output 1:
Enter the expected winner team of IPL Season 4
Chennai Super Kings
Enter the expected runner Team of IPL Season 4
Mumbai Indians
Expected IPL Season 4 winner: Chennai Super Kings
Expected IPL Season 4 runner: Mumbai Indians

Sample Input and Output 1:
Enter the expected winner team of IPL Season 4
Pune Warriors
TeamNameNotFoundException: Entered team is not a part of IPL Season 4

java.lang.*
1. Object
2. Wrapper class
3. String
4. StringBuffer
5. StringBuilder

Java (2 concept)
Error - generated at compile time - syntacial/human made error
Exception - generated at runtime - logical error, eg: Divide by 0, file not found

6. Throwable class
- used to handle exception handling in Java

2 types of Exception
1. Checked Exception
- all subclasses of Exception class excluding RuntimeException. Eventhough we have written 100% correct prg also, ur code has to be surrounded by try/catch block or throws keyword otherwise ur prg will not compile
- Will insist the programmer to surround the code using try/catch block or throws keyword otherwise ur prg will not compile

2. Unchecked Exception
- all subclasses of RuntimeException class
- Eventhough ur prg contains an exception,the compiler will just compile the program but at runtime we get related exception
- Will not insist the programmer to surround the code using try/catch block or throws keyword, the prg will compile but at runtime we get related exception

Constructors
1. Throwable()
2. Throwable(String msg)
3. Throwable(String msg,Throwable t)

Types of Exception
1. ArthimeticException - divide anything by 0z
2. ArrayIndexOutOfBoundsException - only for datatype array
```
int a[]=new int[3];
a[4]=1;
```
3. StringIndexOutOfBoundsException - only for string array
```
String a[]=new String[3];
a[4]="1";
```
4. NegativeArraySizeException
```
int a[]=new int[-3];
```
5. NumberFormatException
```
int a=Integer.parseInt("abc");
```
6. ArrayStoreException
```
int a[]=new int[3];
a[0]="one";
```
7. NullPointerException
```
class A{
  void show(){
  }
}
class Main {PSVM {
  A a=new A();
  a.show();
  a=null;
  a.show();
```
8. ClassCastException
```
class A{
}
class B extends A {}

A a=new A();
B b=new B();
 a=b;  //correct
 b=a;  //Exception
```

5 keywords
1. try
   - Program to be monitored for exception has to be put inside try block
2. catch
   - used to catch the exception generated, mainly used to print user defined messages when an exception occurs
3. finally
   - optional stmt, it will be executed everytime irrespective of exception occurs or not
   - used for closing the resource or cleaning the memory
   - used in 3 cases - file, database, socket programming

```
try {
fp=fopen("a.txt","r");
read operation
}
catch(Exception e) {
}
finally {
  fclose(fp);
}
```

3 ways

```
1. try{                  2. try{              3. try{
}                     }                   }          catch(Exception e){        catch(Exception e){        finally {
}                      }                   }
finally{
}
```

- We should not write anything between trycatch and finally

To get input from user
1. Using Scanner/BufferedReader
2. Directly define in prg itself
3. Using CommandLine argument
   - We give input while running the prg in command line
   - When we use args argument inside the prg, then for that prg we have to give input through command line
   - Each command line should be separated by space

```
public class Main {
        public static void main(String[] args) {
          for(int i=0;i<args.length;i++) {
                System.out.println(args[i]);
          }
  }
}
```

Rigth Click - Run as - Run configuration - Goto Arguments tab - Under Program Arguments we have to give input separated with space

```
public class Main {
        public static void main(String[] args) {
          try {
            int a=Integer.parseInt(args[0]);
            int c=10/a;
            System.out.println(c);
          }
          catch(ArithmeticException e) {
                System.out.println("Divide by 0 "+e);
          }
  }
}
```

Multi catch statement
   - single try can contain multiple catch blocks

```
public class Main {
        public static void main(String[] args) {
          try {
            int a=Integer.parseInt(args[0]);
            int c=10/a;
            System.out.println(c);
            int b[]= {23};
```

```
            b[50]=100;
          }
          catch(ArithmeticException e) {
                System.out.println("Divide by 0 "+e);
          }
          catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Array Index: "+e);
          }

    }
}
```

- Whenever we define general class called Exception or Throwable, it should be always present in last catch block otherwise it leads to compilation error

```
public class Main {
      public static void main(String[] args) {
          try {
            int a=Integer.parseInt(args[0]);
            int c=10/a;
            System.out.println(c);
            int b[]= {23};
            b[50]=100;
          }
          /*catch(/*Exception e*/ Throwable e) {
                System.out.println(e);
          }*/    //Compilation error
          catch(ArithmeticException e) {
                System.out.println("Divide by 0 "+e);
          }
          catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Array Index: "+e);
          }
          catch(NumberFormatException e) {
                System.out.println(e);
          }
          catch(NullPointerException e) {
                System.out.println(e);
          }
          catch(/*Exception e*/ Throwable e) {
                System.out.println(e);
          }


    }
}
```

- From JDK1.7 version, we define multiple exception in a single catch block using |

```
public class Main {
      public static void main(String[] args) {
          try {
            int a=Integer.parseInt(args[0]);
            int c=10/a;
            System.out.println(c);
            int b[]= {23};
            b[50]=100;
          }

          catch(ArithmeticException|ArrayIndexOutOfBoundsException
                        |NumberFormatException|NullPointerException e) {
                System.out.println(e);
          }
    catch(Exception e) {
          System.out.println(e);
    }

    }
}
```

4. throw keyword
   - used to manually throw an exception

- whenever it invokes throw keyword, it will automatically goes to related catch block

Syntax: throw new Exception(String msg);

```
public class Main {
	static void demo() {
		try {
		  throw new NullPointerException("Hello");
		}
		catch(NullPointerException e) {
			System.out.println("Caught");
			throw e;
		}
	}
	public static void main(String[] args) {
		try {
	    demo();
		}
		catch(NullPointerException e) {
			System.out.println("Recaught");
		}
   }
}
```

5. throws keyword
   - throws keyword is used to declare an exception, and used only in methods
   - Used to indicate that the method might throw one of the exception

```
public class Main {
	static void demo() throws NullPointerException {
		  throw new NullPointerException("Hello");
	}
	public static void main(String[] args) {
		try {
	    demo();
		}
		catch(NullPointerException e) {
			System.out.println("Recaught");
		}
   }
}
```

```
public class Main {
	static void demoA() {
		try {
			System.out.println("Inside demoA");
			throw new RuntimeException();
		}
		/*catch(RuntimeException e) {
			System.out.println("Exception caught inside demoA");
		}*/
		finally {
			System.out.println("demoA finally");
		}
	}
	static void demoB() {
		try {
			System.out.println("Inside demoB");
			return;
		}
		finally {
			System.out.println("demoB Finally");
		}
	}
	public static void main(String[] args) {
		try {
		  demoA();
		}
		catch(RuntimeException e) {
			System.out.println("Exception caught");
```

```
                }
                demoB();
    }
}
```

User defined Exceptions
- Your userdefined exception class should extend Exception class and override toString()

```
class A extends Exception {
}

class NotValidAgeException extends Exception{
        String s1="";

        public NotValidAgeException(String s1) {
                this.s1=s1;
        }

        @Override
        public String toString() {
                return s1;
        }

}
public class Main {
        static void validateAge() throws NotValidAgeException {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter age");
                int age=sc.nextInt();
                if(age<18) {
                        throw new NotValidAgeException("Your age is not eligible");
                }
                else {
                        System.out.println("Your age is eligible");
                }
        }
        public static void main(String[] args) {
            try {
                validateAge();
            }
            catch(NotValidAgeException e) {
                System.out.println(e);
            }

    }
}
```

Assertions
- Available from JDK1.5 onwards
- used to check boolean conditions at runtime

Syntax: assert <<expression>>;
assert <<expression>>:String message;
- By default assertion is disabled in Java, while running we have to enable the assertion using -ea options
- Provided with assert keyword, AssertionError class is unchecked exceptio n because it is inherited from Error class
- If assert condition fails then it will throw AssertionError class

```
public class Main {
        static double withdraw(double balance, double amount) {
                assert (balance>=amount);  //Java will just ignore this line
                return (balance-amount);
        }
        public static void main(String[] args) {
            System.out.println(withdraw(1000,500));  //500.0
            System.out.println(withdraw(1000,2000)); //-1000.0
    }
}

public class Main {
        static double withdraw(double balance, double amount) {
                assert (balance>=amount);
                return (balance-amount);
```

```
        }
        public static void main(String[] args) {
            System.out.println(withdraw(1000,500));  //500.0
            System.out.println(withdraw(1000,2000)); //throws AssertionError class
    }
}


public class Main {
        static double withdraw(double balance, double amount) {
                assert (balance>=amount):"Balance is insufficient";
                return (balance-amount);
        }
        public static void main(String[] args) {
            System.out.println(withdraw(1000,500));  //500.0
            System.out.println(withdraw(1000,2000)); //-1000.0
    }
}
```

try with resources
- Available from JDK1.7 onwards
- Used to close the resources automatically at the end of stmt, using AutoCloseable interface

Syntax:

```
  try(resources) {

  }
  catch(Exception e){
  }
```

```
public class Main {
        public static void main(String[] args) {
                String line="";
            try(BufferedReader br=new BufferedReader(new FileReader("C:\\Spring\\JavaExample\\src\\com\\pack\\Main.java"));
                PrintWriter pw=new PrintWriter(new File("a.txt"));){
                while((line=br.readLine())!=null) {
                        System.out.println(line);
                }
            }
            catch(Exception e) {
                System.out.println(e);
            }
    }
}
```

From JDK1.9 onwards, enhancement in try with resources, we can define the resources outside the try block and we can use it

```
public class Main {
        public static void main(String[] args) throws Exception {
                String line="";
                BufferedReader br=new BufferedReader(new FileReader("C:\\Spring\\JavaExample\\src\\com\\pack\\Main.java"));
         PrintWriter pw=new PrintWriter(new File("a.txt"));
            try(br;pw){
                while((line=br.readLine())!=null) {
                        System.out.println(line);
                }
            }
            catch(Exception e) {
                System.out.println(e);
            }
    }
}
```

Exception handling in method overriding
1. If superclass method does not declare an exception , then subclass overridden method cannot declare the checked exception but it can declare unchecked exception

```
class A {
        void add() {

        }
```

```
}
class B extends A  {
        void add() throws ArithmeticException {  //correct

        }
}

class A {
        void add() {

        }
}
class B extends A  {
        void add() throws IOException { //error

        }
}
```

2. If superclass method declares an exception , then subclass overridden method can declare the same exception,subclass exception or no exception but it cannot declare parent exception

```
class A {
        void add() throws Exception {

        }
}
class B extends A  {
        void add() throws Exception {  //correct

        }
}

class A {
        void add() throws Exception {

        }
}
class B extends A  {
        void add() {   //correct

        }
}

class A {
        void add() throws ArithmeticException {

        }
}
class B extends A  {
        void add() throws Exception {  //error

        }
}

class A {
        void add() throws Exception {

        }
}
class B extends A  {
        void add() throws ArithmeticException {  //correct

        }
}
```

Subject: RE: Reg: Day 6 Notes

Classification: **Confidential**

Handson

1. IPL Merchandise is an alternative form of revenue and brings substantial income.

Another merchandise company called “Fundas” has come up with a native flair offering customized mugs, team badges, helmets, etc. The company has decided to print a special code on the merchandises. The special code contains the captions of the IPL teams along with the jersey number of the celebrity players of the corresponding team on the merchandises. The code to be printed is considered valid for printing only if their:
--> first word is one of the team's caption [RCB, MI, CSK, SRH, KXIP, DD, KKR, RPSG and GL]
--> second word is the jersey number
Write a program that reads a string S corresponding to the team name and jersey number and validates if the name is eligible to be printed on the stocks.
Include a class UserMainCode with a static method called validateTeam which accepts a string and its return type is bool. In this method display the details as given in sample input and output.
Create a Class Main which would be used to accept two Strings and call the static method called validateTeam present in UserMainCode.
Input Format:
First line of the input is a string S, that corresponds to the team name and jersey number.

Output Format:
Output should display “Valid” if the team name and jersey number is valid for printing. Otherwise print “Invalid”.

Sample Input 1 :
CSK 7
Sample Output 1 :
Valid

Sample Input 2 :
RCB1 18
Sample Output 2 :
Invalid

2. In Cricket it is so common that the players are often called by their last names. Few cricketers have their origins from the same regions of the country which is why they have common last names. Sandeep is now entrusted with yet another task that is to check for the last names of players.

Given two player's names P1 and P2, Sandeep has to write a program to find if the last name of the two given players are the same. Help him do this using String Builder method.
Include a class UserMainCode with a static method called display which accepts two strings and its return type is void. In this method display the details as given in sample input and output.

Create a Class Main which would be used to accepts two Strings and call the static method called display present in UserMainCode.
Input Format:
First line of the input is a string P1, that corresponds to the first player's name.
Second line of the input is a string P2, that corresponds to the second player's name.

Output Format:
Output should print in a single line “Yes” if the last names of the players are the same. Otherwise print “No”.

Sample input
Mohit Sharma
Rohit Sharma

Sample output
Yes

Sample input
Mohit Sharma
Virat Kohli

Sample output
No

3. Write a program to read a string and find whether it is a valid string or not.
Validation Rules :
1. It accepts the player name as a string and is appended with the player's skill.
2.It ends with a symbol ,the symbol is '*' if the player is a raider and it is '#' if the player is a defender.
3.Player's first name,last name and skill should start with upper case letters

Include a class UserMainCode with a static method validatePlayer which accepts a string. In this method check whether the given player name is valid as per the validation rules mentioned above. The return type is Boolean.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:
Input consists of a string.
Output consists of a string “Valid” or “Invalid”.
Refer sample output for formatting specifications.

Sample Input 1:
Anup Kumar-Raider*
Sample Output 1:
Valid

Sample Input 2:
Surender Nada-Defender#
Sample Output 2:
Valid

Sample Input 3:
Rohit Rana-Defender*
Sample Output 3:
Invalid

4. Write a program to read a string and find whether it is a valid string or not.
Validation Rules :
1. It accepts the player name as a string ,player name length should be between 4 to 20.

2..It starts with four digit number representing the player's debutant year.

3.It ends with single digit number representing the number of IPL season played by the player.

Include a class UserMainCode with a static method validatePlayer which accepts a string. In this method check whether the given player name is valid as per the validation rules mentioned above. The return type is Boolean.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:
Input consists of a string.

Output consists of a string “Valid” or “Invalid”.
Refer sample output for formatting specifications.

Sample Input 1:

2013Mohit Sharma3

Sample Output 1:

Valid

Sample Input 2:

200Mohammad Kaif

Sample Output 2:

Invalid

5. T20 IPL board get the number of player names and finds the players having the string “Sharma” in their name. Write a program to get the player names in an array and use Contains method to get the players whose name has the string "Sharma".

Input and Output Format:
Refer sample input and output for formatting specifications.

Sample Input/Output 1:
Enter number of players
5
Enter player names
Rohit Sharma
Adam Smith

Ishant Sharma
Mohit Sharma
Jaspirit Bumrah
Rohit Sharma
Ishant Sharma
Mohit Sharma

java.lang.* package
1. Object class

2. Wrapper class
- Float class
- Double class

-Integer,Short,Byte,Long wrapper class
Constructors
1. Integer(int i)
Integer i=new Integer(20);
Integer(String s) throws NFE
Integer i=new Integer("20");
Integer i1=new Integer("abc");

2. Short(short s)
Short s1=new Short(23); //error
Short s2=new Short((short)23); //correct
Short(String s) throws NFE
Short s3=new Short("23");

3. Byte(byte b)
Byte b1=new Byte(12); //error
Byte b2=new Byte((byte)12);
Byte(String s) throws NFE
Byte b3=new Byte("12");

4. Long(long l)
Long l1=new Long(14); //correct
Long l1=new Long(14l); //correct
Long(String s) throws NFE
Long l1=new Long("14");

Methods
1. static int parseInt(String s)
2. static int parseInt(String s,int radix)

```
public class Main {
        public static void main(String[] args) {
          String s="42";
          int i=Integer.parseInt(s);
          System.out.println(i);  //42
          int j=Integer.parseInt(s,5);
          //2*5^0=2
          //4*5^1=20
          System.out.println(j); //22
          System.out.println(Integer.toBinaryString(2)); //10
          System.out.println(Integer.toOctalString(9)); //11
          System.out.println(Integer.toHexString(10)); //a
          System.out.println(Integer.MAX_VALUE);
          System.out.println(Integer.MIN_VALUE);
          System.out.println(Integer.TYPE); //int
    }
}
```

5. Boolean class
- used to perform operation on boolean datatype

Constructor
1. Boolean(boolean b)

```
Boolean b1=new Boolean(true);
sop(b1);  //true
Boolean b2=new Boolean(false);
```

```
sop(b2); //false
Boolean b3=new Boolean(True);
sop(b3);  //error
Boolean b4=new Boolean(Helo);
sop(b4);  //error
```

2. Boolean(String s) - we can give only true in any format it will return true, otherwise it returns false

```
Boolean b1=new Boolean("true");
sop(b1);  //true
Boolean b2=new Boolean("false");
sop(b2);  //false
Boolean b3=new Boolean("True");
sop(b3);  //true
Boolean b4=new Boolean("hello");
sop(b4);  //false
Boolean b5=new Boolean("TRUE");
sop(b5); //true
```

6. Character class
   - used to perform operation on char datatype

Constructor
1. Character(char c)

Methods
1. static boolean isDigit(char c)
2. static boolean isLetter(char c)
3. static boolean isLetterOrDigit(char c)
4. static boolean isUpperCase(char c)
5. static boolean isLowerCase(char c)
6. static boolean isSpace(char c)
7. static char toUpperCase(char c)
8. static char toLowerCase(char c)

Autoboxing and Unboxing
- Introduced from JDK1.5 onwards
- Autoboxing is automatic conversion of primitive datatype to wrapper class
- Unboxing is automatic conversion of wrapper class to datatype

```
With autoboxing                    Without autoboxing
Integer i;                         Integer i;
int j;                             int j;
i=25;  //Autoboxing                i=new Integer(25);
j=i;   //Unboxing                  j=i.intValue();
```

3. String class
   - String is final class in java.lang.*
   - Fixed length of char, hence it is called immutable class (ie) we cant increase or decrease its size at runtime

Syntax: public class String extends Object implements Serializable, Cloneable, Comparable, CharSequence

Constructor
1. String()
2. String(String s)
3. String(byte[] b)
4. String(byte[] b, int start,int numofbytes)
5. String(char[] c)
6. String(char[] c, int start,int numofchars)
7. String(StringBuffer s)

Methods
1. String toString() - String representation of an object
2. char charAt(int pos) - return a single char
3. void getChars(int start,int end,char[] buf, int targetstart) - return a group of char
4. byte[] getBytes() - convert string to byte[]
5. char[] toCharArray() - convert String to char[]
6. boolean equals(String s) - check equality of content by taking case into consideration
7. boolean equalsIgnoreCase(String s) - check equality of content by without taking case into consideration
8. ==(equal versus) - check to equality of object reference
9. boolean startsWith(String s)
10. boolean endsWith(String s)

11. String toLowerCase()
12. String toUpperCase()
13. int compareTo(String s) - compare 2 strings taking case into consideration and do sorting, if two values are equal it returns 0, greater means +1, lesser means -1
14. int compareToIgnoreCase(String s) - compare 2 strings without taking case into consideration and do sorting, if two values are equal it returns 0, greater means +1, lesser means -1
15. String intern() - it stores string object info in string pool

String pool is a storage space in the Java heap memory where string literals are stored. It is also known as String Constant Pool or String Intern Pool. It is privately maintained by the Java String class. By default, the String pool is empty.

In stack memory, only the primitive data types like- int, char, byte, short, boolean, long, float and double are stored. Whereas, in the heap memory, non-primitive data types like strings are stored. A reference to this location is held by the stack memory.

The String Pool is empty by default, and it is maintained privately by the String class.
When we create a string literal, the JVM first checks that literal in the String Constant Pool. If the literal is already present in the pool, its reference is stored in the variable.
However, if the string literal is not found, the JVM creates a new string object in the String Constant Pool and returns its reference.

```
String s1="Hello";  //literal - string pool
String s2="Hello";
String s3="hi";

String s4=new String("Gello"); - stored in heap memory
String s5=new String("Gello"); - created new memory in heap
s4.intern();   s4 into string pool
```

16. int indexOf(char c)
    int indexOf(String s)
    int indexOf(int charval,int startindex)
    int indexOf(String s,int index)
      - used to return the position of first occurence of char in given string, if not found it returns -1

17. int lastIndexOf(char c)
    int lastIndexOf(String s)
    int lastIndexOf(String s,int index)
      - used to return the position of last occurence of char in given string, if not found it returns -1
18. String substring(int start) - return part of string from start position
19. String substring(int start,int end) - return part of string from start to end-1
20. int length()
21. String concat(String s) or + operator
22. String trim() - remove leading and trailing space
23. String replace(char old,char new)
24. String[] split(String delimiter)
    String[] split(String delimiter,int limit)
      - used to split the string based on delimiter
25. boolean matches(String regex) - match the string based on regex
26. boolean regionMatches(boolean ignorecase,int toffset,String search,int ooffset,int length) - used to match part of a string with another string
27. int codePointAt(int index)
28. int codePointBefore(int index)
29. int codePointCount(int beginindex, int endindex)
30. boolean contains(String s)
31. static String join(String delimiter,String...s) - used to join the string based on delimiter
32. static String copyValueOf(char[] c)
    static String copyValueOf(char[] c,int offset,int count)
33. static String format(String format,Object val)

```
int a=13;
printf("%d",a);
```

```
%+-0wspecifier.precision
+ - add space before number
- - add space after number
0 - pad with zero
w - total width
specifier - d,f,x,o,s
precision - after decimal place
```

```
public class Main {
        public static void main(String[] args) {
          byte[] b= {65,66,67,68,69};
```

```
String s1=new String(b);
System.out.println(s1); //ABCDE
String s2=new String(b,1,3);
System.out.println(s2); //BCD

char c[]= {'J','a','v','a'};
String s3=new String(c);
System.out.println(s3); //Java
String s4=new String(c,0,2);
System.out.println(s4); //Ja
String s5=new String(s4);
System.out.println(s5); //Ja

String s6="Hello";  //literal
System.out.println(s6.charAt(1));   //e

String s7="This is a demo of getchars method";
int start=10, end=14;
char buf[]=new char[end-start];  //4
s7.getChars(start, end, buf, 0);
System.out.println(buf); //demo

String s8="ABCD";
byte b1[]=s8.getBytes();
for(byte b2:b1)
        System.out.println(b2); //65 66 67 68

char c1[]=s8.toCharArray();
for(char c2:c1)
        System.out.println(c2);  //A B C D

String s9=new String("Hello");
String s10=new String("Hello");
String s11=new String("HELLO");
System.out.println(s9.equals(s10)); //true
System.out.println(s9.equals(s11));  //false
System.out.println(s9.equalsIgnoreCase(s11));  //true
System.out.println(s9==s10);  //false
String s12=s10;
System.out.println(s10==s12); //true

String s13="Hello";
String s14="Hello";
System.out.println(s13==s14); //true

System.out.println("Foobar".startsWith("Foo"));  //true
System.out.println("Foobar".endsWith("bar"));  //true
System.out.println("Foobar".toLowerCase()); //foobar
System.out.println("Foobar".toUpperCase()); //FOOBAR
System.out.println("hello".compareTo("hello")); //0
System.out.println("cat".compareTo("hello")); //-5
System.out.println("hello".compareTo("cat")); //5
System.out.println("hell".compareTo("heck"));  //9

String s15="It is the time for all good men to come to their "
                + "country and pay their due tax";
System.out.println(s15.indexOf('t'));  //1
System.out.println(s15.indexOf("the")); //6
System.out.println(s15.indexOf("THE")); //-1
System.out.println(s15.indexOf(" ")); //2
System.out.println(s15.indexOf(116,7)); //10
System.out.println(s15.indexOf("the",10)); //43
System.out.println(s15.lastIndexOf('t'));  //75
System.out.println(s15.lastIndexOf("the")); //65
System.out.println(s15.lastIndexOf("the",65)); //43

String s16="Helloworld";
System.out.println(s16.substring(6)); //orld
System.out.println(s16.substring(2,6)); //llow - start to end-1

String s17="Hello";
```

```
System.out.println(s17.length());  //5
System.out.println(s17.concat("world"));  //Helloworld
System.out.println(s17.length());  //5
String s18=s17.concat("world");
System.out.println(s17); //Hello
System.out.println(s18); //Helloworld

String s19=" Hello world ";
System.out.println(s19.length()); //13
System.out.println(s19.trim());  //Hello world
System.out.println(s19.length());  //13
String s20=s19.trim();
System.out.println(s20.length());  //11

System.out.println("Hello".replace('e', 'i'));  //Hillo

String s21="one-two-three";
String t1[]=s21.split("-");
for(String t:t1)
        System.out.println(t);  //one   two   three

String s22="one.two.three";
String t2[]=s22.split([file://.]\\.);
for(String t:t2)
        System.out.println(t);  //one    two    three

String s23="A*bunch*of*stars";
String t3[]=s23.split("\\*");
for(String t:t3)
        System.out.println(t);  //A    bunch    of    stars

String s24="A*bunch*of*stars";
String t4[]=s24.split("\\*",3);
for(String t:t4)
        System.out.println(t);  //A    bunch    of*stars

String s25="string is a class";
String t5[]=s25.split("s");
for(String t:t5)
        System.out.println(t);  //tring i
                // a cla

String s26="No concession, no concillation, no compromise and just give and take policy";
String t6[]=s26.split("concession|concillation|compromise|(give and take)");
for(String t:t6)
        System.out.println(t);  //No
                //, no
                //, no
                //and just
                //policy

String s27="Welcome to Java";
System.out.println(s27.matches("(.*) to PHP")); //false
System.out.println(s27.matches("Welcome to (.*)")); //true
System.out.println(s27.matches("Java"));  //false
System.out.println(s27.matches("(.*) to (.*)")); //true

String s28="ABC Windows test"; //Windows test
System.out.println(s28.regionMatches( 4, "windows", 0, 7));  //false
System.out.println(s28.regionMatches(true,4, "windows", 0, 7)); //true

System.out.println("abcd".codePointAt(0)); //97
System.out.println("abcd".codePointBefore(2)); //98
System.out.println("abcdefg".codePointCount(0, 5));  //5
// System.out.println("abcdefg".codePointCount(0, 9));  //exception
System.out.println("Hello".contains("e"));  //true
System.out.println("Hello".contains("E"));  //false

System.out.println(String.join("-", "one","two","three")); //one-two-three

char c2[]= {'a','b','c','d','e','f','g'};
```

```
        String s29="";
        System.out.println(s29.copyValueOf(c2)); //abcdefg
        System.out.println(s29.copyValueOf(c2, 3, 2)); //de

        int a=35;
        System.out.println(a);  //35
        System.out.println(String.format("%d", a)); //35
        System.out.println(String.format("|%d|", a)); //|35|
        System.out.println(String.format("|%5d|", a)); //5- total width |   35|
        System.out.println(String.format("|%-5d|", a)); //|35   |
        System.out.println(String.format("|%06d|", a));  //|000035|
        //System.out.println(String.format("|%-06d|", a)); //exception
        System.out.println(String.format("|%f|", 123.456)); //|123.456000|
        System.out.println(String.format("|%.2f|", 123.456)); //|123.46|
        System.out.println(String.format("|%8.2f|", 123.456)); //|   123.46|
    }
}
```

4. StringBuffer class
   - present in java.lang.*
   - It is a mutable class, variable length of char (ie) we can increase or decrease its size at runtime
   - By default StringBuffer is synchronzied or thread safe, it always gives lower performance
   - Default capacity of StringBuffer is 16
   - We cant override equals() in StringBuffer, using toString() we convert StringBuffer to String class and then we apply equals() - it returns false

Constructors
1. StringBuffer()
2. StringBuffer(String s)
3. StringBuffer(int capacity)

Methods
1. int length()
2. char charAt(int pos)
3. void setCharAt(int index,char c)
4. void setLength(int length)
5. int capacity()
6. StringBuffer append(int i)
   StringBuffer append(char c)
   StringBuffer append(String s)
      - add at the end of Stringbuffer
7. StringBuffer insert(int index,int i)
   StringBuffer insert(int index,char i)
   StringBuffer insert(int index,String i)
    - insert anywhere in StringBuffer
8. StringBuffer reverse()
9. StringBuffer replace(int start,int end, String s) //start to end-1
10. StringBuffer delete(int start,int end) - delete group of char
11. StringBuffer deleteCharAt(int index) - delete single char

```
public class Main {
        public static void main(String[] args) {
          StringBuffer sb1=new StringBuffer("Hello");
          System.out.println(sb1);  //Hello
          System.out.println(sb1.length()); //5
          System.out.println(sb1.capacity());  //21 = 16+5
          System.out.println(sb1.charAt(1));  //e
          sb1.setCharAt(1, 'i');
          System.out.println(sb1);  //Hillo
          sb1.setLength(2);
          System.out.println(sb1); //Hi
          System.out.println(sb1.length()); //2

          int b=10;
          StringBuffer sb2=new StringBuffer();
          String s=sb2.append("a=").append(b).append("!").toString();
          System.out.println(s); //a=10!

          StringBuffer sb3=new StringBuffer("I Java");
          System.out.println(sb3.insert(2, "like ")); //I like Java

          StringBuffer sb4=new StringBuffer("Hello");
```

```
        System.out.println(sb4.reverse());  //olleH

        StringBuffer sb5=new StringBuffer("This is a test");
        System.out.println(sb5.replace(5, 7, "was")); //This was a test

        StringBuffer sb6=new StringBuffer("This is a test");
        System.out.println(sb6.delete(5, 7));  //This a test
        System.out.println(sb6.deleteCharAt(0));  //his a test
    }
}
```

5. StringBuilder class
   - Available from JDK1.5 onwards
   - It is similar to StringBuffer but it is not synchronized or threadsafe so it gives better performance

---

Subject: RE: Reg: Day 5 Notes

Classification: **Confidential**

Handson

1. Team Details
Create a class named Team with the following private member variables / attributes

String name;
String coach;
String location;
String players
String captain

Include appropriate getters and setters.
[Naming Convention:
getters : getName getCoach ..
setters : setName, setCoach...]

Include a 5-argument argument constructor in this class. The arguments passed to the constructor are in this order --- name, coach, location,players,captain.
Include a default empty constructor.

Create another class named Main and include a main method to test the above class(Print the output in Main Class).

Input and Output Format:
Refer sample input and output for formatting specifications.
Read the team details as a string value separate by '#'. Use string.split() function to display the team details
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the team details
CSK#Stephen Fleming#Chennai#MS Dhoni,Aswin,Raina,Hussey,Maxwel,Bravo,Morkel,Jadeja,Mohit Sharma,Hayden,du plessis,Abhinav Mukund#MS Dhoni
Team : CSK
Coach : Stephen Fleming
Location : Chennai
Players : MS Dhoni,Aswin,Raina,Hussey,Maxwel,Bravo,Morkel,Jadeja,Mohit Sharma,Hayden,du plessis,Abhinav Mukund
Captain : MS Dhoni

2. Wicket details

Create a class named Wicket with the following private member variables / attributes

Long over
Long ball
String wicketType
String playerName
String bowlerName

Include a 5-argument argument constructor in this class. The arguments passed to the constructor are in this order --- over, ball,wicketType,playerName,bowlerName.
Include a default empty constructor.

Include appropriate getters and setters.
[Naming Convention:
getters : getOver getBall ...
setters : setOver, setBall...]

Create another class named Main and include a main method to test the above class(Print the output in Main Class).

Input and Output Format:
Refer sample input and output for formatting specifications.
Use array of objects to read wicket details and use String.split() function to display the wicket details.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the number of wickets
3
Enter the details of wicket 1
5,2,LBW,Gautam Gambir,Aswin
Enter the details of wicket 2
6,6,Bowled,Brad Hogg,Dwayne Bravo
Enter the details of wicket 3
7,3,Stumped,Robin Uthappa,Ravindra Jadeja
Wicket Details
Over : 5
Ball : 2
Wicket Type : LBW
Player Name : Gautam Gambir
Bowler Name : Aswin
Over : 6
Ball : 6
Wicket Type : Bowled
Player Name : Brad Hogg
Bowler Name : Dwayne Bravo
Over : 7
Ball : 3
Wicket Type : Stumped
Player Name : Robin Uthappa
Bowler Name : Ravindra Jadeja

Predefined Packages
1. java.lang.* - Language Package - only one optional pkg

1. Object class
    - super class of all the classes (ie) any predefined or userdefined class by default inherit Object class

void add(int a) - int as arg
void add(String s)- String as arg
void add(Test t) - object of only Test class as arg
void add(Employee e) - object of only Employee class as arg
void add(Object o) - object of any class as arg

Constructor
- Object()

Methods
1. String toString() - String representation of an object

String s1=new String("Hello"); //all predefined class by default contains toString(), so if we print object of any predefined class it will print the content instead of object reference
sop(s1); //Hello

Sample s2=new Sample("Hello"); //Userdefined
sop(s2); //object reference

Whenever we print object of any class it will print only memory reference. But we dont print memory reference instead we want to print some content then in that case we have to override toString()

```
class A {
	String s1;
	A(String s){
		s1=s;
	}
	@Override
	public String toString() {
		return s1;
	}

}
public class Main {
	public static void main(String[] args) {
	   String a1=new String("Hello");
	   System.out.println(a1);   //Hello
	   A a2=new A("Hello");
	   //System.out.println(a2); //object reference
	   System.out.println(a2);  //automatically invokes toString() - Hello
	   System.out.println(a2.toString()); //Hello
	}

}
```

2. boolean equals(Object o) - used to check equality of the content
3. == (equals versus) - used to check equality of object reference

If we are not using new operator, then == also works like equals()

```
public class Main {
	public static void main(String[] args) {
	Integer a1=new Integer(42);
	Integer a2=new Integer(42);
	System.out.println(a1.equals(a2));  //true
	   System.out.println(a1==a2);  //false
	   String s1=new String("Hello");
	   String s2=new String("Hello");

	   System.out.println(s1.equals(s2));  //true
	   System.out.println(s1==s2);  //false
	   String s3=s2;
	   System.out.println(s2.equals(s3)); //true
	   System.out.println(s2==s3);  //true
	   String s4="Hello";  //literal
	   String s5="Hello";  //literal
	   System.out.println(s4.equals(s5));  //true
	   System.out.println(s4==s5);  //true
	}
}
```

4. int hashCode()
   - When we store some value into object, internally an address will be generated for that value, to return that address we use hashCode()
   - If two objects are equal according to equals(), then hashCode will return same

```
public class Main {
	public static void main(String[] args) {
	   String s1=new String("Hello");
	   String s2=new String("Hello");
	   System.out.println(s1.hashCode());
	   System.out.println(s2.hashCode());
	}
}
```

5. protected void finalize()

Box b=new Box(); real object stored in heap
reference(stack)
b=null;

Explicitly Garbage Collection
1. System.gc()

2. Runtime.gc()

finalize() used to perform cleanup activity before destroying the object. It is called by garbage collector before destroying the object from memory and invoked only once in the lifecycle of program

Unreferencinng the object
1. By anonymous object
Anonymous object are those object which are created without reference variable

```
new Box().add();
```

2. By nulling reference

```
Box b=new Box();
b=null;
```

3. By assigning the reference to another variable

```
Box b1=new Box();  //1000
Box b2=new Box();  //2000
b1=b2;
sop(b1);  //2000
```

1. finalize() of which class is going to be called

```
public class Main {
	public static void main(String[] args) {
	  String s1=new String("Hello");
	  s1=null;
	  System.gc();
	  System.out.println("Garbage Collection is done");
	}
	protected void finalize() {
		System.out.println("Cleanup is done");
	}
}
```

```
public class Main {
	public static void main(String[] args) {
	  Main s1=new Main();
	  s1=null;
	  System.gc();
	  System.out.println("Garbage Collection is done");
	}
	protected void finalize() {
		System.out.println("Cleanup is done");
	}
}
```

2. Explicit call to finalize

```
public class Main {
	public static void main(String[] args) {
	  Main s1=new Main();
	  Main s2=new Main(); //2000
	  s1=s2;
	  s1.finalize();   //not destroy the object
	  System.out.println("Garbage Collection is done");
	  System.gc();
	}
	protected void finalize() {

		System.out.println("Cleanup is done");
	}
}
```

6. final void wait()
7. final void wait(long msec)
8. final void notify()
9. final void notifyAll()

10. Object clone()
- used to create exact copy of an object
- clone() always follow shallow copy
- If we want to take copy of an object using clone(), then that class should implements Cloneable interface which is a marker interface, in case if it is not implemented Cloneable interface then it will raise CloneNotSupportedException

Shallow copy
- will have exact copy of all fields of original object (ie) any changes made to those object through clone object will be reflected in original object and vice versa

```
class Address {
	String city;
	String state;
	String country;
	public Address(String city, String state, String country) {
		super();
		this.city = city;
		this.state = state;
		this.country = country;
	}
}
class Employee implements Cloneable {
	int empid;
	String name;
	Address address;
	public Employee(int empid, String name, Address address) {
		super();
		this.empid = empid;
		this.name = name;
		this.address = address;
	}

	protected Object clone() throws CloneNotSupportedException {
		return super.clone();
	}
}
public class Main {
	public static void main(String[] args) {
	   Address addr=new Address("Chennai","Tamilnadu","India");
	   Employee e1=new Employee(10,"Ram",addr);
	   Employee e2=null;
	   try {
			e2=(Employee)e1.clone();
		} catch (CloneNotSupportedException e) {
			e.printStackTrace();
		}
	   System.out.println(e1.address.state);  //Tamilnadu
	   e2.address.state="Kerela";
	   System.out.println(e1.address.state); //Kerela
	}
}
```

2. Wrapper class
- classes that supports the primitive datatype, to perform any operation on datatype
- All wrapper classes are immutable class(u cant change)

```
int a=10;
```

datatype wrapper class
int - Integer
byte - Byte
short - Short
long - Long
boolean - Boolean
char - Character
float - Float
double - Double

1. Float class
- used to perform operation on float datatype

Constructor
1. Float(float f)
   Float f1=new Float(3.14f);
2. Float(double d)
   Float f2=new Float(3.14);
3. Float(String s) throws NumberFormatException
   Float f3=new Float("3.14");
   Float f4=new Float("1bc"); //NumberFormatException

```
int a=10, b=10;
if(a==b)

float f1=3.14f,f2=3.14f;
if(f1==f2)  //error
```

Methods
1. static int compare(float f1,float f2) - compare 2 float datatype, if equal it return 0, if greater 1, if lesser -1
2. int compareTo(Float f) - compare 2 Float object , if equal it return 0, if greater 1, if lesser -1
3. static boolean isInfinite() - check float datatype is infinite or not
4. boolean isInfinite() - check Float object is infinite or not
5. static boolean isFinite() - check float datatype is finite or not
6. boolean isFinite() - check Float object is finite or not
7. static boolean isNaN() - check float datatype is nan or not
8. boolean isNaN() - check Float object is nan or not
9. static float parseFloat(String s) throws NFE - convert String to related datatype
10. static Float valueOf(String s) throws NFE - convert String to Float object
11. float floatValue() - convert Float object to float datatype
12. int intValue()
13. double doubleValue()
14. short shortValue()
15. long longValue()

Constants
1. Float.MAX_VALUE
2. Float.MIN_VALUE
3. Float.POSITIVE_INFINITY
4. Float.NEGATIVE_INFINITY
5. Float.TYPE

```
public class Main {
	public static void main(String[] args) {
	  float f1=3.14f;
	  float f2=3.14f;
	  System.out.println(Float.compare(f1,f2));  //0

	  Float f3=new Float(3.14f);
	  Float f4=new Float(3.14f);
	  System.out.println(f3.compareTo(f4)); //0

	  float f5=(float)(1/0.);
	  System.out.println(Float.isInfinite(f5));  //true
	  Float f6=new Float(1/0.);
	  System.out.println(f6.isInfinite()); //true

	  float f7=(float)Math.sqrt(-4);
	  System.out.println(Float.isNaN(f7));  //true
	  Float f8=new Float(Math.sqrt(-4));
	  System.out.println(f8.isNaN()); //true

	  String s="3.14";
	  float f9=Float.parseFloat(s);
	  System.out.println(f9); //3.14
	  Float f10=Float.valueOf(s);
	  System.out.println(f10);  //3.14
	  float f11=f10.floatValue();
	  System.out.println(f11); //3.14
	  int i=f10.intValue();
	  System.out.println(i);  //3

	  System.out.println(Float.MAX_VALUE);
	  System.out.println(Float.MIN_VALUE);
```

```
        System.out.println(Float.POSITIVE_INFINITY);
        System.out.println(Float.NEGATIVE_INFINITY);
        System.out.println(Float.TYPE);
    }
}
```

2. Double class
   - used to perform operation on double datatype

Constructor
1. Double(double d)
2. Double(String s) throws NFE

Subject: RE: Reg: Day 4 Notes

Classification: **Confidential**

Handson
1. Abstract Class I – Shape

[Adhere to the OOPs specifications specified here. Follow the naming conventions for getters and setters. Download the template code provided and fill in the missing code]

Create an abstract class named Shape with the following protected attributes / member variables.

String name

Include a 1-argument constructor.

Include getters and setters.

Include an abstract method named calculateArea() . This method returns a Float value.

Create a class named Circle . The class Circle is a derived class of Shape. Include the following private attributes / member variables.

Integer radius

Include a 2-argument constructor. The order of the arguments is name, radius.

Include getters and setters.

Override the abstract method calculateArea() defined in the Shape class. This method returns the area of the circle. [Take the value of pi as 3.14]

Create a class named Square . The class Square is a derived class of Shape. Include the following private attributes / member variables.

Integer side

Include a 2-argument constructor. The order of the arguments is name, side.

Include getters and setters.

Override the abstract method calculateArea() defined in the Shape class. This method returns the area of the square.

Create a class named Rectangle . The class Rectangle is a derived class of Shape. Include the following private attributes / member variables.

Integer length

Integer breadth

Include a 3-argument constructor. The order of the arguments is name, length, breadth

Include getters and setters.

Override the abstract method calculateArea() defined in the Shape class. This method returns the area of the rectangle.

Create another class called Main. In the method, create instances of the above classes and test the above classes.

Input and Output Format:

Refer sample input and output for formatting specifications.

All Float values are displayed correct to 2 decimal places.

All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output 1:

Circle

Square

Rectangle

Enter the shape name

Circle

Enter the radius

25

Area of Circle is 1962.50

Sample Input and Output 2:

Circle

Square

Rectangle

Enter the shape name

Square

Enter the side

23

Area of Square is 529.0

Sample Input and Output 3:

Circle

Square

Rectangle

Enter the shape name

Rectangle

Enter the length

45

Enter the breadth

60

Area of Rectangle is 2700.00

2.Abstract Class II – Card

[Adhere to the OOPs specifications specified here. Follow the naming conventions for getters and setters. Download the template code provided and fill in the missing code]

Create an abstract class named Card with the following protected attributes / member variables.

String holderName;

String cardNumber;

String expiryDate;

Include appropriate getters and setters.

Include appropriate constructors. In the 3-argument constructor, the order of the arguments is holderName, cardNumber, expiryDate.

Create a class named MembershipCard. The class MembershipCard is a derived class of Card. Include the following private attributes / member variables.

Integer rating

Include appropriate getters and setters.

Include appropriate constructors. In the 4-argument constructor, the order of the arguments is holderName, cardNumber, expiryDate, rating.

Create a class named PaybackCard. The class PaybackCard is a derived class of Card. Include the following private attributes / member variables.

Integer pointsEarned;

Double totalAmount;

Include appropriate getters and setters.

Include appropriate constructors. In the 5-argument constructor, the order of the arguments is holderName, cardNumber, expiryDate, pointsEarned, totalAmount.

Create another class called Main. In the method, create instances of the above classes and test the above classes. The card details are entered separated by a '|'.

Input and Output Format:

Refer sample input and output for formatting specifications.

All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output 1:

Select the Card

1.Payback Card

2.Membership Card

1

Enter the Card Details:

Anandhi|12345|14/01/2020

Enter points in card

1000

Enter Amount

50000

Anandhi's Payback Card Details:

Card Number 12345

Points Earned 1000

Total Amount 50000.0

Sample Input and Output 2:

Select the Card

1.Payback Card

2.Membership Card

2

Enter the Card Details:

Collin|45678|20/11/2021

Enter rating in card

10

Collin's Membership Card Details:

Card Number 45678

Rating 10

3. Simple Interface

An Interface consists of a contract of set of methods with only declaration and no implementation. Any class which implements the interface commits that all methods would be implemented.
Here is a program to illustrate a simple interface.

1. Create an interface IPlayerStatistics
Add a method with the following prototype
--- public void displayPlayerStatistics
2. Create class Player which implements the IPlayerStatistics Interface
Include private data members :
String name
String teamName
Integer noOfMatches
Long totalRunsScored
Integer noOfWicketsTaken

Include an 5 argument constructor with following arguments: name, teamName, noOfMatches, totalRunsScored, noOfWicketsTaken
and implement the interface method public void displayPlayerStatistics to display the player details.

Create a Main class with main method to test test above classes.

Sample Input and Output 1:
[All text in bold corresponds to input and the rest corresponds to output.]

Enter player name
Ravichandran Ashwin
Enter team name

Chennai Super Kings
Enter number of matches played
86
Enter total runs scored
185
Enter number of wickets taken
89
Player Details
Player name : Ravichandran Ashwin
Team name : Chennai Super Kings
No of matches : 86
Total runsscored : 185
No of wickets taken : 89

4.Interface II

1. Create an interface IPlayerStatistics
Add a method with the following prototype
public void displayPlayerStatistics
2. Create base class CricketPlayer
Include private data members : name, teamName, noOfMatches.
Include an 3 argument constructor with following arguments: name, teamName, noOfMatches.

3. Create derived class Bowler which extends CricketPlayer and implements the interface IplayerStatistics
Include private data members : noOfWickets.
Include an 1 argument constructor with the argument: noOfWickets and implement the interface method public void displayPlayerStatistics to display the player details.

4. Create derived class Batsman which extends CricketPlayer and implements the interface IPlayerStatistics
Include private data members : runs.
Include an 1 argument constructor with the argument: runs and implement the interface method public void displayPlayerStatistics to display the player details.

5. Create derived class WicketKeeper which extends CricketPlayer and implements the interface IPlayerStatistics
Include private data members : noOfCatches, noOfStumpings ,runs, noOfDismissals .
Include an 4 argument constructor with following arguments: noOfCatches, noOfStumpings,runs, noOfDismissals and implement the interface method public void displayPlayerStatistics to display the player details.

6. Create derived class AllRounder which extends CricketPlayer and implements the interface IPlayerStatistics
Include private data members : runs,noOfWickets.
Include an 2 argument constructor with following arguments: runs, noOfWickets and implement the interface method public void displayPlayerStatistics to display the player details.

Create a Main class with main method to test test above classes.

Input and Output Format:
Refer sample input and output for format specifications.
[All text in bold corresponds to input and the rest corresponds to output.]

Sample Input and Output 1:
Menu
1.Bowler
2.Batsman
3.WicketKeeper
4.AllRounder
Enter your choice
1
Enter the Bowler details
Enter player name
Ravichandran Ashwin
Enter team name
Chennai Super Kings
Enter number of matches played
111
Enter number of wickets taken
100
Player name : Ravichandran Ashwin
Team name : Chennai Super Kings
No of matches : 111
No of wickets taken : 100

Do you want to continue?
YES
Menu
1.Bowler
2.Batsman
3.WicketKeeper
4.AllRounder
Enter your choice
4
Enter the AllRounder details
Enter player name
Shane Watson
Enter team name
Royal Challengers Bangalore
Enter number of matches played
94
Enter the runs scored
2551
Enter number of wickets taken
81
Player name : Shane Watson
Team name : Royal Challengers Bangalore
No of matches : 94
Runs scored : 2551
No of wickets taken : 81
Do you want to continue?
NO

Sample Input and output 2:
Menu
1.Bowler
2.Batsman
3.WicketKeeper
4.AllRounder
Enter your choice
2
Enter the Batsman details
Enter player name
Virat Kohli
Enter team name
Royal Challengers Bangalore
Enter number of matches played
139
Enter the runs scored
4110
Player name : Virat Kohli
Team name : Royal Challengers Bangalore
No of matches : 139
Runs scored : 4110
Do you want to continue?
YES
Menu
1.Bowler
2.Batsman
3.WicketKeeper
4.AllRounder
Enter your choice
3
Enter the WicketKeeper details
Enter player name
Mahendra Singh Dhoni
Enter team name
Chennai Super Kings
Enter number of matches played
143
Enter number of catches taken
52
Enter number of stumpings
22
Enter number of dismissals
74

Enter the runs scored
3271
Player name : Mahendra Singh Dhoni
Team name : Chennai Super Kings
No of matches : 143
No of catches taken : 52
No of stumpings : 22
No of dismissals : 74
Runs scored : 3271
Do you want to continue?
NO

Problem Statement:
What is the output of the following program?

```
class Eggs {
   int doX(Long x, Long y) { return 1;}
   int doX(long... x) { return 2;}
   int doX(Integer x, Integer y) { return 3; }
   int doX(Number n, Number m) { return 4; }

   public static void main(String[ ] args) {
      new Eggs().go();
   }

   void go() {
      short s = 7;
      System.out.println(doX(s,s) + " ");
      System.out.println(doX(7,7));
   }
}
```

A. 1 1
B. 2 1
C. 3 1
D. 4 1
E. 2 3
F. 3 3
G. 4 3

Problem Statement:
What is the output of the following program?

```
class Mixer {
   Mixer() { }
   Mixer(Mixer m) { m1 = m; }
   Mixer m1;
   public static void main(String[ ] args) {
      Mixer m2 = new Mixer();
      Mixer m3 = new Mixer(m2);
      m3.go();
      Mixer m4 = m3.m1;
      m4.go();
      Mixer m5 = m2.m1;
      m5.go();
   }
   void go() {
      System.out.println("hi ");
   }
}
```

Problem Statement:
What is the output of the following program?

```
class Bird {
   { System.out.print("b1 "); }

   public Bird() {
      System.out.print("b2 ");
```

```
  }
}

class Raptor extends Bird {
  static { System.out.print("r1 "); }

  public Raptor() {
    System.out.print("r2 ");
  }

  { System.out.print("r3 "); }

  static { System.out.print("r4 "); }
}

class Hawk extends Raptor {
  public static void main(String[] args) {
    System.out.print("pre ");
    new Hawk();
    System.out.println("hawk ");
  }
}
```

A. pre b1 b2 r3 r2 hawk
B. pre b2 b1 r2 r3 hawk
C. pre b2 b1 r2 r3 hawk r1 r4
D. r1 r4 pre b1 b2 r3 r2 hawk
E. r1 r4 pre b2 b1 r2 r3 hawk
F. pre r1 r4 b1 b2 r3 r2 hawk
G. pre r1 r4 b2 b1 r2 r3 hawk
H. The order of output cannot be predicted.
I. Compilation fails.

Problem Statement:
What is the output of the following program?

```
class Clidders {
  public final void flipper() {
    System.out.println("Clidder");
  }
}

public class Clidlets  extends Clidders {
  public void flipper() {
    System.out.println("Flip a Clidlet");
    super.flipper();
  }
  public static void main(String[] args) {
    new Clidlets().flipper();
  }
}
```

A. Flip a Clidlet
B. Flip a Clidder
C. Flip a Clidder
Flip a Clidlet
D. Flip a Clidlet
Flip a Clidder
E. Compilation fails.

Problem Statement:
Given the following:

```
 i.  interface Base {
ii.     boolean m1 ();
iii.    byte m2(short s);
iv.  }
```

Which code fragments will compile? (Choose all that apply.)

a) interface Base2 implements Base { }
b) abstract class Class2 extends Base {
public boolean m1 () { return true; } }
c) abstract class Class2 implements Base { }
d) abstract class Class2 implements Base {
public boolean m1 () { return (true); } }
e) class Class2 implements Base {
boolean m1 () { return false; }
byte m2 (short s) { return 42; } }

Problem Statement:
Which of the following declare a compatible abstract class? (Choose all that apply.)

a) public abstract class Canine { public Bark speak(); }
b) public abstract class Canine { public Bark speak() { } }
c) public class Canine { public abstract Bark speak(); }
d) public class Canine abstract { public abstract Bark speak(); }

Problem Statement:
Given:
public abstract interface Frobnicate { public void twiddle(String s); }

Which is a correct class?

a) public abstract void twiddle(String s) { }
}
b) public abstract class Frob implements Frobnicate { }
c) public class Frob extends Frobnicate {
public void twiddle(Integer i) { }
}
d) public class Frob implements Frobnicate {
public void twiddle(Integer i) { }
}
e) public class Frob implements Frobnicate {
public void twiddle(String i) { }
public void twiddle(Integer s) { }
}

Problem Statement:
Given:
1. class Zing {
2. protected Hmpf h;
3. }
4. class Woop extends Zing { }
5. class Hmpf { }

Which is true?
A. Woop IS-A Hmpf and HAS-A zing.
B. Zing IS-A Woop and HAS-A Hmpf.
C. Hmpf HAS-A Woop and Woop IS-A Zing.
D. Woop HAS-A Hmpf and Woop IS-A Zing.
E. Zing HAS-A Hmpf and Zing IS-A Woop.

Problem Statement:
Given:
public class MyOuter {
public static class MyInner {
public static void foo()) { }
}
}
Which, if placed in a class other than MyOuter or MyInner, instantiates an instance of the nested class?
A. MyOuter.MyInner m = new MyOuter.MyInner();
B. MyOuter.MyInner m2 = new MyInner();
C. MyOuter m = new MyOuter();
MyOuter.MyInner mi = m.new MyOuter.MyInner();

D. MyInner mi = new MyOuter.MyInner();

Problem Statement:
What is the output of the following program?

```
public abstract class AbstractTest {

   public int getNum()) {
      return 45;
   }
   public abstract class Bar {
      public int getNum()) {
         return 38;
      }
  }
   public static void main()String[] args) {
      AbstractTest t = new AbstractTest()) {
         public int getNum()) {
            return 22;
         }
      };
      AbstractTest.Bar f = t.new Bar()) {
         public int getNum()) {
            return 57;
         }
      };
      System.out.println()f.getNum()) + " " + t.getNum()));
   }
}
```

A. 57 22
B. 45 38
C. 45 57
D. An exception occurs at runtime.
E. Compilation fails.

super keyword
- It is used to access ur base class constructor, base class method and base class variable

```
class A {
        int i;
        int j;

        A(int a,int b){
                i=a;
                j=b;
        }

        void show() {
                System.out.println("i= "+i+" j= "+j);
        }
}
class B extends A {
        int k;

        B(int a,int b,int c){
                super(a,b);
                k=c;
        }
        void show() {
           super.show();
                System.out.println("k= "+k);
        }
}
public class Main {
        public static void main(String[] args) {
                B b=new B(1,2,3);
                b.show();
        }
```

```
}
```

Method overriding
- same method name, same number,order and datatype of arg, same return type of method and present in different class and the class should be inherited

No Constructor overriding

Dynamic Method Dispatch
In order to have effective method overriding, we go for Dynamic Method dispatch
- We always create an object for base class but we store the reference of derived class but not vice versa, so the compiler thinks that object is created for base class but only at runtime it comes to know it contains the reference of derived class, in this way we achieve runtime polymorphism

```
class Sphere {
	void volume() {
		System.out.println("Sphere volume");
	}
}
class Hemisphere extends Sphere {
	void volume() {
		System.out.println("Hemisphere volume");
	}
}
public class Main {
	public static void main(String[] args) {
		/*Hemisphere h=new Hemisphere();
		h.volume();  //Hemisphere volume
		Sphere s=new Sphere();
		s.volume();*/ //Sphere volume

		Sphere s=new Hemisphere();  //DMD
		s.volume();  //Hemisphere volume
		s=new Sphere();  //Hemisphere reference garbage collected and s contain sphere reference
		s.volume(); //Sphere volume
	}
}

//Hiearchial inheritance
class A {
	void callback() {
		System.out.println("A's callback implementation");
	}
}
class B extends A {
	void callback() {
		System.out.println("B's callback implementation");
	}
}
class C extends A {
	void callback() {
		System.out.println("C's callback implementation");
	}
}
public class Main {
	public static void main(String[] args) {
		A a=new A();
		a.callback();
		a=new B();  //DMD
		a.callback();
		a=new C();
		a.callback();
	}
}
```

How constructors are invoked in inheritance?
- Invoked as top down approach

```
//Multilevel inheritance
class A {
	A() {
```

```
                System.out.println("1");
        }
}
class B extends A {
        B() {
                System.out.println("2");
        }
}
class C extends B{
        C(){
                System.out.println("3");
        }
}
public class Main {
        public static void main(String[] args) {
                C c=new C();  //123
        }
}

//Multilevel inheritance
class A {
        A(int a) {
                System.out.println("1");
        }
}
class B extends A {
        B() {
                System.out.println("2");
        }
}
class C extends B{
        C(){
                System.out.println("3");
        }
}
public class Main {
        public static void main(String[] args) {
                C c=new C();  //compilation error
        }
}
```

First it will go to the related constructor (ie)into C class constructor and check whether we have used super() or this(), then it will work according to that, if it is not then by default it will invoke only default constructor of base class

```
//Multilevel inheritance
class A {
        A(int a) {
                System.out.println("1");
        }
    /*A() {
       System.out.println("4");
    }*/
}
class B extends A {
        B() {
          super(10);
                System.out.println("2");
        }
}
class C extends B{
        C(){
                System.out.println("3");
        }
}
public class Main {
        public static void main(String[] args) {
                C c=new C();   //123
        }
}
```

final keyword

- It is a non access specifier
- When a class is declared as final, inheritance is not possible

```
final class A {
	A() {
		System.out.println("1");
	}
}
class B extends A {  //error
	B() {
		System.out.println("2");
	}
}
```

- When a method is declared as final, method overriding is not possible

```
class A {
	final void add() {
		System.out.println("1");
	}
}
class B extends A {
	void add() {  //error
		System.out.println("2");
	}
}
```

- When a variable is declared as final, it will acts like constant

Relationship - 2 types

1. is a relationship - when we perform inheritance

```
class Animal{
}
class Cat extends Animal{
}
```

Cat is a Animal

2. has-a relationship - Composition - one class contains another class

```
class Wheel{
}
class Car{
  Wheel w=new Wheel();    //Car has a Wheel
}
```

Covariant return type

- Before JDK1.5, it is not possible to override any method by changing its return type
- From JDK1.5, it is possible to override method by changing its return type (ie) only non primitive (ie) object
- Avoid type casting, wrong typecasting which leads to ClassCastException at runtime

```
class A {
	A show() {
		return this;
	}
	void print() {
		System.out.println("Inside A's method");
	}
}
class B extends A {
	B show(){
		return this;
	}
	void print() {
		System.out.println("Inside's B method");
	}
}
class C extends B {
	C show(){
		return this;
	}
	void print() {
		System.out.println("Inside's B method");
	}
```

```
}
public class Main {
	public static void main(String[] args) {
		A a1=new A();
		a1.show().print();  //Inside A's method
		B b1=new B();
		b1.show().print();  //Inside B's method
	}
}
```

Compound block/Instance block
- It will be invoked before the constructor invokes

Syntax:
```
{
  //logic
}
```

only in case inheritance, first all static block will be executed from top to bottom

18 21 19 1 6 11 15 17 5 7 2 4 3 8 10 9 14 16 12 13 20

```
class A {
	static {   //static block
		System.out.println("1");
	}
	A(){
		System.out.println("2");
	}
	A(String s){
		this(10);
		System.out.println("3");
	}
	A(int a){
		this();
		System.out.println("4");
	}
	{   //Compound block
		System.out.println("5");
	}
	static {
		System.out.println("6");
	}
	{
		System.out.println("7");
	}
}
class B extends A {
	{
		System.out.println("8");
	}
	B(){
		super("hello");
		System.out.println("9");
	}
	{
		System.out.println("10");
	}
	static {
		System.out.println("11");
	}
}
class C extends B {
	C(){
		System.out.println("12");
	}
	C(int a){
		this();
		System.out.println("13");
	}
	{
		System.out.println("14");
```

```
	}
	static {
		System.out.println("15");
	}
	{
		System.out.println("16");
	}
	static {
		System.out.println("17");
	}
}
public class Main {
	static {
		System.out.println("18");
	}
	public static void main(String[] args) {
	   System.out.println("19");
	   C c=new C(10);
	   System.out.println("20");
	}
	static {
		System.out.println("21");
	}
}
```

abstract keyword
- It is a non access specifier
- When a class is declared as abstract, u cannot create an object for that class
- When a method is declared as abstract, it does not contain any defination, it just ends with semicolon
- Variables cant be declared as abstract
- When a class contains abstract method then that class should be declared as abstract otherwise error, but not necessarlly all abstract class should contain abstract methods
- Abstract class can also non abstract method (normal method)
- Abstract class can also be inherited, so when u inherit an abstract class then complusorly we have to provide the defination of abstract method in the inherited class or if we not defining the defination then we have to make inherited class itself as abstract
- Abstract class contain constrcutor but we cant invoke it

```
abstract class A {
	abstract void show();
	void method() {
		System.out.println("Concrete implementation");
	}
}
/*abstract*/ class B extends A {
	void show() {
		System.out.println("Abstract method");
	}
}
public class Main {
	public static void main(String[] args) {
	  B b=new B();
	  b.show();
	  b.method();
	}

}
```

Anonmyous Inner class
- used to access normal method of abstract class without inheritance

```
abstract class A {
	abstract void show();
	void add() {
		System.out.println("Concrete implementation");
	}
}

public class Main {
	public static void main(String[] args) {
	   A a=new A() {   //AIC
		public void show() {
```

```
                        System.out.println("abstract method");
                }
                public void add() {
                        System.out.println("hello");
                }
          };
          a.show();
          a.add();
        }

}
```

Interface
- No multiple inheritance instead we use interface

```
class A extends B,C,D{   //error
}
```

- Interface are syntacially similar to classes but contains only method declaration and variable declaration and initialization

Syntax:

```
<<accessspecifier>>  interface  interfacename {
    //method declaration
    //variable declaration and initialization
}
```

- By default all interfaces are abstract, so we cant create an object for interface
- By default all interface methods are public and abstract, so interface method dosent contain any defination it just ends with semicolon
- By default all interface variables are public, final, static, so we can access interface variable using "interfacename.variablename"
- Using "implements" keyword we can use interface in class
- When a class implements an interface then complusorly we have to provide the defination of interface method in the class with public access specifier or define the class itself to be abstract
- The implemented class can also contain non interface method
- Interface can also be inherited

1 class extends 1 class
1 class implement n interface
1 interface extends n interface

- Marker interface - interface which dosent contain anything like Cloneable, Serializable, Remote
- From JDK1.8 onwards, interface can also contain default and static method
- From JDK1.9 onwards, interface can also contain private and private static methods

```
interface Maths {
        void arthimetic(int a,int b);
        int c=10;
}
/*abstract*/ class A implements Maths {
        @Override
        public void arthimetic(int a, int b) {
                System.out.println(a+b);
                System.out.println(Maths.c);
        }
}
class B implements Maths {
        @Override
        public void arthimetic(int a, int b) {
                System.out.println(a-b);
        }
}
public class Main {
        public static void main(String[] args) {
          /* A a=new A();
          a.arthimetic(5, 3);  //8 10
          B b=new B();
          b.arthimetic(5, 1); */ //4

                Maths m=new A();  //DMD
                m.arthimetic(5, 3);  //8  10
                m=new B();
                m.arthimetic(5, 1);  //4
        }

}
```

```
interface A {
	void meth1();
}
interface B {
	void meth2();
}
interface C extends A,B{
	void meth3();
}
class Example implements C {

	@Override
	public void meth1() {
		System.out.println("Meth1");
	}

	@Override
	public void meth2() {
		System.out.println("Meth2");
	}

	@Override
	public void meth3() {
		System.out.println("Meth3");
	}

}
public class Main {
	public static void main(String[] args) {
	  Example e=new Example();
	  e.meth1();
	  e.meth2();
	  e.meth3();
	}
}
```

Subject: RE: Reg: Day 3 Notes

Classification: **Confidential**

Handson

1. Create a class named Venue with the following member variables / attributes (Default access)

Data Type	Variable Name
String	name
String	city

Create another class called Main and write a main method to test the above class.

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the venue name
M. A. Chidambaram Stadium
Enter the city name
Chennai
Venue Details :
Venue Name : M. A. Chidambaram Stadium
City Name : Chennai

2. Create a class named Player with the following member variables / attributes (Default access)

Data Type	Variable Name
String	name
String	country

String skill

Create another class named Main and write a main method to test the above class.

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the player name
MS Dhoni
Enter the country name
India
Enter the skill
All Rounder
Player Details :
Player Name : MS Dhoni
Country Name : India
Skill : All Rounder

3. Create a class named Delivery with the following public member variables / attributes

Data Type	Variable Name
Long	over
Long	ball
Long	runs
String	batsman
String	bowler
String	nonStriker

Include a method in the class named displayDeliveryDetails(). In this method, display the details of the delivery in the format shown in the sample output. This method does not accept any arguments and its return type is void.

Create another class called Main and write a main method to test the above class.

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the over
1
Enter the ball
1
Enter the runs
4
Enter the batsman name
MS Dhoni
Enter the bowler name
Dale steyn
Enter the nonStriker name
Suresh Raina
Delivery Details :
Over : 1
Ball : 1
Runs : 4
Batsman : MS Dhoni
Bowler : Dale steyn
NonStriker : Suresh Raina

4. Create a class named Player with the following member variables / attributes (default access)

Data Type	Variable Name
String	name
String	country
String	skill

Create another class called Main and write a main method to get the player details in a string seperated by comma.Use String. split() function to display the details.

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the player details
MS Dhoni,India,All Rounder
Player Details
Player Name : MS Dhoni
Country Name : India
Skill : All Rounder

5. Create a class named Venue with the following private member variables / attributes
String name
String city

Include appropriate getters and setters.
[Naming Convention:
getters : getName getCity...
setters : setName setCity...]

Create another class and write a main method to test the above class. In the main method, get the choice from the user and update the corresponding venue details.

Input and Output Format:
Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.

Sample Input and Output :
Enter the venue name
Green Park Stadium
Enter the city name
Kanpur
Venue Details
Venue Name : Green Park Stadium
City Name : Kanpur
Verify and Update Venue Details
Menu
1.Update Venue Name
2.Update City Name
3.All informations Correct/Exit
Type 1 or 2 or 3
2
Enter the city name
Chennai
Venue Details
Venue Name : Green Park Stadium
City Name : Chennai
Verify and Update Venue Details
Menu
1.Update Venue Name
2.Update City Name
3.All informations Correct/Exit
Type 1 or 2 or 3
1
Enter the venue name
Chidambaram Stadium
Venue Details
Venue Name : Chidambaram Stadium
City Name : Chennai
Verify and Update Venue Details
Menu
1.Update Venue Name
2.Update City Name
3.All informations Correct/Exit
Type 1 or 2 or 3
3
Venue Details
Venue Name : Chidambaram Stadium

City Name : Chennai

6. Venue Details

[Note :
Strictly adhere to the object oriented specifications given as a part of the problem statement.
Use the same class names and member variable names.
Follow the naming conventions mentioned for getters / setters.
Create 3 separate classes in 3 separate files.]

Create a class named Venue with the following private member variables / attributes

Data Type	Variable Name
String	name
String	city

Include appropriate getters, setters and constructors.
Naming Convention :
getters --- getName,etc...
setters --- setName,etc...

Include a default constructor.
Include a 2-argument constructor --- the 1st argument corresponds to the name and the 2nd argument corresponds to the city.

Override the toString() method to display the venue details in the following format specified in the output.

Create a class named VenueBO and include the following methods

No	Method Name	Method Description
1	void displayVenueDetails(Venue venue)	In this method, display the details of the venue.

Input and Output Format:

Refer sample input and output for formatting specifications.
All text in bold corresponds to input and the rest corresponds to output.
Note : The statement " Venue Details" in the output is displayed in the method inside the BO class.
Sample Input and Output :
Enter the venue name
M Chidhambaram Stadium
Enter the city name
Chennai
Venue Details
M Chidhambaram Stadium,Chennai

this() constructor
- used to invoke different constructor of the same class, it should be always present only in the first line otherwise it gives compilation error

```
class A {
	A(){
		System.out.println("1");
	}
	A(int a){
		this("hello");
		System.out.println("2");
	}
	A(String s){
		this();
		System.out.println("3");
	}
}
public class Main  {
	public static void main(String[] args)  {
		//A a1=new A();
		A a2=new A(10); //1 3 2
		//A a3=new A("hello");
	}
}
```

```
class A {
```

```
	A(){
		this(5);
		System.out.println("1");
	}
	A(int a){
		this("hello");
		System.out.println("2");
	}
	A(String s){
		System.out.println("3");
	}
}
public class Main  {
	public static void main(String[] args)  {
		A a1=new A();
		//A a2=new A(10); //3 2 1
		//A a3=new A("hello");
	}
}
```

Polymorphism - one class which takes many forms - 2 types

1. static/compile time polymorphism
2. Dynamic/runtime polymorphism

Static/Compile time polymorphism
- Using overloading
- 2 types - Method overloading and Constructor overloading

Method overloading
- same method name, but different number, order and datatype of argument, different return type but present in same class

```
class OverloadDemo {
	//Method overloading
	void test() {
		System.out.println("No parameter");
	}
	/*void test(int a) {
		System.out.println("a = "+a);
	}*/
	void test(int a,int b) {
		System.out.println("a = "+a+" b= "+b);
	}
	double test(double a) {
		System.out.println("a = "+a);
		return a*a;
	}
}
public class Main  {
	public static void main(String[] args)  {
	  OverloadDemo ob=new OverloadDemo();
	  ob.test();
	  ob.test(10);  //implicit conversion
	  ob.test(10,20);
	  double r=ob.test(10.34);
	  System.out.println(r);
	}
}
```

Constructor overloading
- same constructor name, but different number, order and datatype of argument, but present in same class

```
class Box {
	double width;
	double depth;
	double height;

	double volume() {
		return width*depth*height;
	}

	//Constructor Overloading
```

```
	Box(){
		width=-1;
		depth=-1;
		height=-1;
	}
	Box(double width,double depth,double height){
		this.width=width;
		this.depth=depth;
		this.height=height;
	}
	Box(double len){
		width=depth=height=len;
	}
}
public class Main  {
	public static void main(String[] args)  {
	   Box b1=new Box();
	   Box b2=new Box(10,20,15);
	   Box b3=new Box(7);
	   double vol;
	   vol=b1.volume();
	   System.out.println("Volume is "+vol); //Volume is -1
	   vol=b2.volume();
	   System.out.println("Volume is "+vol); //Volume is 3000.0
	   vol=b3.volume();
	   System.out.println("Volume is "+vol); //Volume is 343.0
	}
}
```

Passing an object as an argument

```
void add(int a){}
void add(Test t){}
void add(Sample s){}

class Box {
	double width;
	double depth;
	double height;

	double volume() {
		return width*depth*height;
	}

	//Constructor Overloading
	Box(){
		width=-1;
		depth=-1;
		height=-1;
	}
	Box(double width,double depth,double height){
		this.width=width;
		this.depth=depth;
		this.height=height;
	}
	Box(double len){
		width=depth=height=len;
	}
	Box(Box b){
		width=b.width;
		depth=b.depth;
		height=b.height;
	}
}
public class Main  {
	public static void main(String[] args)  {
	   Box b1=new Box();
	   Box b2=new Box(10,20,15);
	   Box b3=new Box(7);
	   Box b4=new Box(b2);
```

```
        double vol;
        vol=b1.volume();
        System.out.println("Volume is "+vol); //Volume is -1
        vol=b2.volume();
        System.out.println("Volume is "+vol); //Volume is 3000.0
        vol=b3.volume();
        System.out.println("Volume is "+vol); //Volume is 343.0
        vol=b4.volume();
        System.out.println("Volume is "+vol); //Volume is 3000.0
    }
}

class Example {
    void add(int a,int b){
      sop(a+b);
    }
    void add(int a,int b, int c){
      sop(a+b+c);
    }
    void add(int a,int b,int c, int d){
      sop(a+b+c+d);
    }
    void add(int a,int b,int c, int d,int e){
      sop(a+b+c+d+e);
    }
}
```

var args
- used to define variable number of argument, 0 to any
- Available from JDK1.5 onwards
- denoted by ...
- If we are passing combination of arg with var arg, then var arg should be always present only as last arg
- One method can have only one var args and that also should be present as last argument

```
void add(int...a){  //0 or any number of int arg

}
void add(String...s){}  //0 or any number of string arg

void add(double...d,String s){}  //error

void add(int a,String...s,float...f){}  //error

class Demo {
    void add(int...a) {
        for(int a1:a)
            System.out.println(a1);
    }
    void add(boolean...b) {
        for(boolean b1:b)
            System.out.println(b1);
    }
    void add(String s,double...d) {
        System.out.println(s);
        for(double d1:d)
            System.out.println(d1);
    }
}
public class Main  {
    public static void main(String...args)  {
      Demo d=new Demo();
      //d.add();
      d.add(2,3,4,5);
      d.add(1,2,3,4,5,6,7,8,9);
      d.add(true,false,true,true,false);
      d.add("hello",1.2,3.4,5.6);
    }
}
```

Access specifiers - public, private, protected, default - can be applied for all (ie) class, interface, method, variables, constants

Non access specifiers - It has its own restrictions (ie) cannot be applied for everything

static keyword
- It is a non access specifier
- Whenever a class is declared as static, there is no need to create an object for that class, outer class cannot be declared as static only inner class can be static

```
class A  {  //outer class
  class B {  //inner class
  }
}
```

- Whenever a method is declared as static, if it is present in same class then we can access using "methodname", if it is present in different class then we can access using "classname.methodname"
- Whenever a variable is declared as static it will acts like global variable, if it is present in same class then we can access using "variablename", if it is present in different class then we can access using "classname.variablename"
- static method can access only static content, if it is not static it is accessed by creating an object

```
class Sample {
  int a;
  public static void main(String[] args){
    System.out.println(a);   //compilation error,static method can access only static content
  }
}
```

```
class Sample {
  static int a;
  boolean b;
  public static void main(String[] args){
    System.out.println(a);  //0
    Sample s=new Sample();
    System.out.println(s.b); //false
  }
}
```

- When we want to execute any task before main(), then we can go for static block which will be executed before main() method

Syntax:

```
static {
  //stmt
}
```

- static method can override only another static method

```
public class Main  {
        static int a=4;
        static int b;

        static void method(int x) {
                System.out.println(x); //10
                System.out.println(a); //4
                System.out.println(b); //8
        }

        static {
                System.out.println("Static block initialized");
                b=a*2;
        }
        public static void main(String...args)  {
            method(10);
        }
}
```

```
class Sample {
        static int a=10;
        static int b=15;
        static void add() {
                System.out.println(a);
        }
}
public class Main  {
        public static void main(String...args)  {
```

```
            Sample.add(); //10
            System.out.println(Sample.b); //15
        }
}
```

System.out.println()/System.in

System is a predefined class

out is an object of PrintStream class and it is static variable in System class

println() method of PrintStream class

```
public class System {
    static PrintStream out;
    static InputStream in;
}
```

static import
- Available from JDK1.5 onwards, only for static method and static variables
- Normally we can invoke static method and static variable using classname.methodname and classname.variablename, but if we want to invoke static method and static variable without classname.methodname and classname.variablename then we can use static import

```
public class Main  {
        public static void main(String...args)  {
          double d=Math.sqrt(16);
          System.out.println(d);
        }
}
```

```
import static java.lang.System.out;
import static java.lang.Math.sqrt;

public class Main  {
        public static void main(String...args)  {
          double d=sqrt(16);
          out.println(d);
        }
}
```

Nested class and Inner class
- A class inside another class

```
//Nested class
class A  {
  class B {
  }
}
```
- Inside class can be either static or non-static

```
class A  {
  class B { //inner class - non static inner class
  }
}

class A  {
  static class B { //static class
  }
}
```
- Outer class properties can be accessed inside inner class only if it is instance variable or should be declared as final, otherwise it is an error
- We cant access inner class properties in outer class

```
class Outer{
        int outer=10;   //Instance variable

        void test() {
                Inner i=new Inner();
                i.display();
        }
        class Inner {   //Inner class
                int y=5;
```

```
                void display() {
                        System.out.println(y+" "+outer);
                }
        }

}
public class Main  {
        public static void main(String...args)  {
          Outer o=new Outer();
          o.test();
        }
}


class Outer{
        int outer=10;   //Instance variable

        class Inner {   //Inner class
                int y=5;
                void display() {
                        System.out.println(y+" "+outer);
                }
        }

}
public class Main  {
        public static void main(String...args)  {
          Outer.Inner o=new Outer().new Inner();
          o.display();
        }
}


class Outer{
        static int outer=10;   //Instance variable

        static class Inner {   //static class
                int y=5;
                void display() {
                        System.out.println(y+" "+outer);
                }
        }

}
public class Main  {
        public static void main(String...args)  {
          Outer.Inner o=new Outer.Inner();
          o.display();
        }
}
```

Inheritance

- accessing the properties of one class into another class if it is not present
- code reusability
- using "extends" keyword
- single, multilevel, hybrid, hiearchial
- No multiple inheritance

```
class Parent {  //Parent/Base/Super class
}
class Child extends Parent {  //child/derived/sub class
}

//Base class - Single inheritance
class Box {
        double width;
        double depth;
        double height;

        double volume() {
                return width*depth*height;
```

```
        }

        Box() {
            width=-1;
            depth=-1;
            height=-1;
        }
        Box(double w,double d,double h){
            width=w;
            depth=d;
            height=h;
        }
        Box(double len){
            width=depth=height=len;
        }
        Box(Box ob){
            width=ob.width;
            depth=ob.depth;
            height=ob.height;
        }
}
//Derived class
class BoxWeight extends Box {
        double weight;

        BoxWeight(){
            width=-1;
            depth=-1;
            height=-1;
            weight=-1;
        }
        BoxWeight(double w,double d,double h,double we){
            width=w;
            depth=d;
            height=h;
            weight=we;
        }
        BoxWeight(double len,double w){
            width=depth=height=len;
            weight=w;
        }
        BoxWeight(BoxWeight ob){
            width=ob.width;
            depth=ob.depth;
            height=ob.height;
            weight=ob.weight;
        }

}
public class Main  {
        public static void main(String...args)  {
            BoxWeight b1=new BoxWeight();
            double vol;
            vol=b1.volume();
            System.out.println("Volume is "+vol); //Volume is -1.0
            System.out.println("Weight is "+b1.weight);

            BoxWeight b2=new BoxWeight(10,20,15,3);
            vol=b2.volume();
            System.out.println("Volume is "+vol); //Volume is 3000.0
            System.out.println("Weight is "+b2.weight); //3

            BoxWeight b3=new BoxWeight(10,20);
            vol=b3.volume();
            System.out.println("Volume is "+vol); //Volume is 1000.0
            System.out.println("Weight is "+b3.weight); //20

            BoxWeight b4=new BoxWeight(b1);
            vol=b4.volume();
            System.out.println("Volume is "+vol); //Volume is -1.0
            System.out.println("Weight is "+b4.weight); //-1
```

```
        }
}
```

super keyword
- used to access base class constructor, base class method and base class variable.
- Only in the case accessing base class constructor, super keyword should be present in first line
- super() and this() cant be used simultaneously

```
//Base class - Single inheritance
class Box {
        private double width;
        private double depth;
        private double height;

        double volume() {
                return width*depth*height;
        }

        Box() {
                width=-1;
                depth=-1;
                height=-1;
        }
        Box(double w,double d,double h){
                width=w;
                depth=d;
                height=h;
        }
        Box(double len){
                width=depth=height=len;
        }
        Box(Box ob){
                width=ob.width;
                depth=ob.depth;
                height=ob.height;
        }
}
//Derived class
class BoxWeight extends Box {
        double weight;

        BoxWeight(){
                super();
                weight=-1;
        }
        BoxWeight(double w,double d,double h,double we){
                super(w,d,h);
                weight=we;
        }
        BoxWeight(double len,double w){
                super(len);
                weight=w;
        }
        BoxWeight(BoxWeight ob){
                super(ob);
                weight=ob.weight;
        }

}
public class Main  {
        public static void main(String...args)  {
            BoxWeight b1=new BoxWeight();
            double vol;
            vol=b1.volume();
            System.out.println("Volume is "+vol); //Volume is -1.0
            System.out.println("Weight is "+b1.weight); //-1

            BoxWeight b2=new BoxWeight(10,20,15,3);
            vol=b2.volume();
            System.out.println("Volume is "+vol); //Volume is 3000.0
            System.out.println("Weight is "+b2.weight); //3
```

```
        BoxWeight b3=new BoxWeight(10,20);
        vol=b3.volume();
        System.out.println("Volume is "+vol); //Volume is 1000.0
        System.out.println("Weight is "+b3.weight); //20

        BoxWeight b4=new BoxWeight(b1);
        vol=b4.volume();
        System.out.println("Volume is "+vol); //Volume is -1.0
        System.out.println("Weight is "+b4.weight); //-1
    }
}

//Base class - Multilevel inheritance
class Box {
    private double width;
    private double depth;
    private double height;

    double volume() {
        return width*depth*height;
    }

    Box() {
        width=-1;
        depth=-1;
        height=-1;
    }
    Box(double w,double d,double h){
        width=w;
        depth=d;
        height=h;
    }
    Box(double len){
        width=depth=height=len;
    }
    Box(Box ob){
        width=ob.width;
        depth=ob.depth;
        height=ob.height;
    }
}
//Derived class
class BoxWeight extends Box {
    double weight;

    BoxWeight(){
        super();
        weight=-1;
    }
    BoxWeight(double w,double d,double h,double we){
        super(w,d,h);
        weight=we;
    }
    BoxWeight(double len,double w){
        super(len);
        weight=w;
    }
    BoxWeight(BoxWeight ob){
        super(ob);
        weight=ob.weight;
    }

}
class Shipment extends BoxWeight {
    double cost;

     Shipment(double w,double d,double h,double we,double c){
        super(w,d,h,we);
        cost=c;
    }
```

```
}
public class Main  {
	public static void main(String...args)  {
	   Shipment s1=new Shipment(10,20,15,3,4);
	   double vol;
	   vol=s1.volume();
	   System.out.println("Volume is "+vol); //3000.0
	   System.out.println("Weight is "+s1.weight);  //3
	   System.out.println("Cost is "+s1.cost); //4
	}
}

class A {
	int i;
}
class B extends A {
	int i;

	B(int i,int j){
		this.i=i;   //from B class
		super.i=j; //from A class
	}
	void show() {
		System.out.println(i+" "+super.i);
	}
}
public class Main  {
	public static void main(String...args)  {
	  B b=new B(1,2);
	  b.show(); // 1  2
	}
}
```

Scanner class
- used to get input from user based on the datatype
- present in java.util.*
- If we use combination of numbers and String, either u can use dummy line called sc.nextLine() or get all input in string (ie) using nextLine() and later convert to its related datatype

int nextInt()
float nextFloat()
double nextDouble()
short nextShort()
byte nextByte()
long nextLong()
boolean nextBoolean()
String nextLine() - read words with space
String next() - read single word without space

```
public class Main  {
	public static void main(String...args)  {
	  Scanner sc=new Scanner(System.in);
	  System.out.println("Enter age");
	  int s2=Integer.parseInt(sc.nextLine());
	  //sc.nextLine();
	  System.out.println("Enter name");
	  String s1=sc.nextLine();

	  //Student st=new Student(s1,s2);
	  Student st=new Student();
	  st.setName(s1);
	  st.setAge(s2);
	  System.out.println(st.getName()+" "+st.getAge());
	}
}
```

Subject: RE: Reg: Day 2 Notes

Classification: **Confidential**

4 J's

JDK - Java Development Kit - JDK is JRE + extra tools that are needed to develop java program such as Java compiler, Java interpreter

JVM - Java Virtual Machine - JVM is like motor of car. It is a software that reads and execute the java bytecode

JIT - Just in time compiler - JVM contains JIT, which translate Java byte code into native machine code for the CPU that the prg is running

JRE - Java Runtime Environment - JRE is JVM and everything that is needed to execute java prg by supporting standard libraries, JRE is complete car

Execution Control statement in Java - 3 types
1. Conditional stmt - 2 types
    a.if/if-else/if-else-if/nested if - check for boolean condition

In C and C++, other than 0 if we provide +ve or -ve value it will return true otherwise false

```
if(10){   //error
  sop("yes");
}
else
  sop("no")

int a=10;
if(a=11){   //error

}
if(a==11){  //correct

}
```

  b. switch case - check multiple condition

Syntax:

```
    switch(expr) {
       case arg:
         //stmt
         break;
       --
       --
       default:
         //stmt;
         break;
    }
```

- expr can be either int/byte/short/char/enum/String(JDK1.7)
- arg should be always final (ie) acts like constant

```
int a=1;
switch(a) {
  case 1:
    SOP("1");
    break;
  case 1:   //error
```

- default stmt can be present anywhere inside switch case
- If we want to come out from switch we have to use break stmt, dont ever use continue inside switch case, it leads compilation error

2. Looping statement

1. for loop

```
     for(initialization;condition;inc/dec){
     }
```

2. do while

```
     do {
        //stmt
```

```
    }
    while(condition);
3  while
    while(condition);
```

3. Flow breaking stmt
1. break - used to stop entire iteration
2. continue - stop the current iteration
3. return - transfer the call back to calling prg

```
public class Main  {
	public static void main(String[] args)  {
		if(10==10) {
			System.out.println("Hello");
		}
		if(true)
			System.out.println("hi");

		int a=10;
		if(a>10)
			System.out.println("yes");
		else
			System.out.println("no");

		switch(10) {
		case 10:
			System.out.println("10");
			//break;
		default:
			System.out.println("30");
			//break;
		case 20:
			System.out.println("20");
			//break;
		}

		char c='b';
		switch(c) {
		case 'b':
			System.out.println("b");
			//break;
		default:
			System.out.println("nothing");
			break;
		case 'a':
			System.out.println("a");
			//break;
		}

		String s1="three";
		switch(s1) {
		case "one":
			System.out.println("10");
			//break;
		default:
			System.out.println("30");
			//break;
		case "three":
			System.out.println("20");
			//break;
		}

		for(int i=0;i<10;i++) {
			if(i%2==0)
				//break;
				continue;
			System.out.println(i+" "); // 1 3 5 7 9
		}
	}
}
```

Array
- collection of similar datatypes

1. int a[]={1,2,3}; //correct - declaring and initializing the values to the array and its size is fixed depending upon the number of elements, so size=3
2. int a[3]={1,2,3}; //error - if we want to provide the size of array then complusorly we have to new operator
3. int a[]=new int[3]; //correct - declare an array of size 3, memory allocated is 12 bytes(3*4)
4. int[] a=new int[3]; //correct
5. int a[]=new int[]{1,2,3}; //correct - Anonmyous array where we can declare and initialize in same line
6. int a[]=new int[3]{1,2,3}; //error - In anonmyous array u should not specify its size
7. int a[]=new int[-3]; //correct - In Java we can declare an array with negative size but at runtime we get NegativeArraySizeException
8. int a[]=new int[3]; //declaring an array of size 3
   a[0]=1; a[1]=2; a[2]=3; //initalizing an array
9. int a[]={1,2,3}; //
   sop(a); //memory reference
10. We want to access individual elements of array - 2 ways

arrayname.length - used to find length of array

i - loopcounter variable

1. Normal for loop

```
int a[]={1,2,3};
int i;
for(i=0;i<a.length;i++) {
   System.out.println(a[i]);  //1 2 3
}

double d[]={1.2,3.4,5.6};
for(int a1=0;a1<d.length;a1++){
 System.out.println(d[a1]);   //1.2   3.4  5.6
}
```

2. for each stmt - available from JDK1.5 onwards, used to access individual elements of array

Syntax: for(countervariable: array){

      }
1. counter variable should be declared only inside foreach stmt, and it should be same datatype as a array
2. 1D array has to put inside a variable, 2D array has to put inside 1D, 3D has to put inside 2D etc

```
int a[]={1,2,3};
for(int w: a)
  System.out.println(w);  // 1  2  3

double d[]={1.2,3.4,5.6};
for(double d1:d)
  System.out.println(d1); //1.2   3.4   5.6
```

1. int a[][]={{1,2},{3,4}}; //correct - declaring and initialize an array 16 bytes(2*2*4)
2. int a[2][2]={{1,2},{3,4}}; //error
3. int a[][]=new int[3][3]; //correct - 36 bytes
4. int[][] a=new int[3][3]; //correct
5. int[] a[]=new int[3][3]; //correct
6. int[][] a=new int[5][5]; //correct - 100 bytes
7. int a[][]=new int[5][5]; //correct - 100 bytes, only 8 bytes remaining 92 bytes will be wasted
    a[0][0]=0;
    a[1][0]=2;
8. int a[][]=new int[5][]; //Correct - Arrays of Array - we have to declare only rows, based on requirment we can allocate ur columns, allocates 16 bytes
   a[0]=new int[1]; a[0][0]=1;
   a[1]=new int[2]; a[1][0]=2; a[1][1]=3;
   a[2]=new int[1]; a[2][0]=4;
9. int a[][]=new int[][5]; //error

```
float f1[][]={{1.2f,3.4f},{4.5f,5.6f}};
sop(f1);  //reference

for(int i=0;i<2;i++) {
  for(int j=0;i<2;j++) {
```

```
    System.out.println(f1[i][j]);   //1.2  3.4  4.5  5.6
  }
}

for(float f2[]:f1)
  for(float f3:f2)
    System.out.println(f3);   //1.2  3.4  4.5  5.6
```

2 types of classes
1. Main class
   - entry point of program, there will be always only one mainclass
   - If it is contain public static void main(String[] args)
   - used to create an object for base class and finally we print the output

2. Base class
   - we have to write all ur logic inside base class, there may be multiple base class

sysout - press ctrl+spacebar
type main - press ctrl+spacebar

ctrl A - Ctrl+shift+F - formatting

class and object

```
//Base class
class Box {
        //Instance variable
        double width;
        double depth;
        double height;
}
public class Main  {
        public static void main(String[] args)  {
                Box b1=new Box();  //created an object called "b1" which contains memory reference and invoke default constructor
                b1.width=10;
                b1.depth=20;
                b1.height=15;

                Box b2=new Box();
                b2.width=3;
                b2.depth=6;
                b2.height=9;

                double vol;  //local variable
                vol=b1.width*b1.depth*b1.height;
                System.out.println("Volume is "+vol); //Volume is 3000.0
                vol=b2.width*b2.depth*b2.height;
                System.out.println("Volume is "+vol); //Volume is 162.0


        }
}
```

Method - used to write any logic

Syntax: returntype methodname(argument){
        //logic
      }

```
//Base class
class Box {
        //Instance variable
        double width;
        double depth;
        double height;

        void volume() {
                System.out.println(width*depth*height);
        }
}
```

```
public class Main  {
	public static void main(String[] args)  {
		Box b1=new Box();  //created an object called "b1" which contains memory reference and invoke default constructor
		b1.width=10;
		b1.depth=20;
		b1.height=15;

		Box b2=new Box();
		b2.width=3;
		b2.depth=6;
		b2.height=9;

		b1.volume();  //3000.0
		b2.volume();  //162.0

	}
}


//Base class
class Box {
	//Instance variable
	double width;
	double depth;
	double height;

	double volume() {
		return (width*depth*height);
	}
}
public class Main  {
	public static void main(String[] args)  {
		Box b1=new Box();  //created an object called "b1" which contains memory reference and invoke default constructor
		b1.width=10;
		b1.depth=20;
		b1.height=15;

		Box b2=new Box();
		b2.width=3;
		b2.depth=6;
		b2.height=9;
		double vol;
		vol=b1.volume();
		System.out.println("Volume is "+vol);
		vol=b2.volume();
		System.out.println("Volume is "+vol);
	}
}
```

Parameterized Method - method that takes an parameters

```
//Base class
class Box {
	//Instance variable
	double width;
	double depth;
	double height;

	double volume() {
		return (width*depth*height);
	}

	void setDim(double w,double d,double h) {
		width=w;
		depth=d;
		height=h;
	}
}
public class Main  {
	public static void main(String[] args)  {
		Box b1=new Box();  //created an object called "b1" which contains memory reference and invoke default constructor
```

```
        b1.setDim(10, 20, 15);
                Box b2=new Box();
                b2.setDim(3,6,9);
                double vol;
                vol=b1.volume();
                System.out.println("Volume is "+vol);
                vol=b2.volume();
                System.out.println("Volume is "+vol);

        }
}
```

2 types of methods
1. Mutator method - setter method - used to set the value - setSize(), setAge()
2. Accessor method - getter method - used to return the value - getSize(), getAge()

Constructor
- What is constructor?
  - Whenever we create an object for a class, memory will be allocated and allocation of memory is called constructor
- When constructor invoked?
  - Whenever we create an object for a class, the constructor will be invoked
- Why constructor?
  - used to initialize values to the variable at time of object creation
- By default all class will contain one default constructor (ie) constuctor without any parameter, and we can create our own constructor where ur class name and constructor name should be same, at that time the already present default constructor will be overlap
- Constructor can be created with or without parameters
- Constructor can have access specifiers
- Constructor should not have return type

```
//Base class
class Box {
        //Instance variable
        double width;
        double depth;
        double height;

        double volume() {
                return (width*depth*height);
        }

        Box(){
                width=-1;
                depth=-1;
                height=-1;
        }
}
public class Main  {
        public static void main(String[] args)  {
                Box b1=new Box();  //created an object called "b1" which contains memory reference and invoke default constructor
                Box b2=new Box();

                double vol;
                vol=b1.volume();
                System.out.println("Volume is "+vol); //Volume is -1.0
                vol=b2.volume();
                System.out.println("Volume is "+vol); //Volume is -1.0

        }
}
```

Parameterized Constructor - constructor which takes parameters

```
//Base class
class Box {
        //Instance variable
        double width;
        double depth;
        double height;

        double volume() {
                return (width*depth*height);
```

```
        }

        Box(double w,double d,double h){   //parameterized constructor
                width=w;
                depth=d;
                height=h;
        }
}
public class Main  {
        public static void main(String[] args)  {
                Box b1=new Box(10,20,15);  //created an object called "b1" which contains memory reference and invoke default constructor
                Box b2=new Box(1,2,3);

                double vol;
                vol=b1.volume();
                System.out.println("Volume is "+vol); //Volume is 3000.0
                vol=b2.volume();
                System.out.println("Volume is "+vol); //Volume is 6.0

        }
}
```

this keyword
- used to refer current class variable
- It is used if ur parameter name and instance variable name are same

```
//Base class
class Box {
        //Instance variable
        double width;
        double depth;
        double height;

        double volume() {
                return (width*depth*height);
        }

        Box(double width,double depth,double height){   //parameterized constructor
                this.width=width;
                this.depth=depth;
                this.height=height;
        }
}
```

Subject: Reg: Day 1 Notes

Classification: **Confidential**

Problem Statement:
Find out whether the following file will compile. If it does not compile, how you would fix it?

```
public static void main(String[ ] args) {
    int x = 5;
    while (x > 1) {
      x = x + 1;
      if (x < 3) {
          System.out.println(“small x”);
      }
    }
}
```

Problem Statement:
Find out whether the following file will compile. If it does not compile, how you would fix it?

```
                                    class Digit {
    public static void main(String[ ] args) {
                        int x = 1;
                        while (x < 10) {
```

```
                    if (x > 3) {
                        System.out.println("big x");
                    }
                }
            }
        }
    }
```

Problem Statement:
Find out whether the following file will compile. If it does not compile, how you would fix it?

```
                                    class Loop {
        public static void main(String[ ] args) {
         int x = 5;
                while (x  > 1) {
                    x = x - 1;
            if (x < 3) {
                System.out.println("small x");
            }
         }
      }
}
```

Problem Statement:
What is the output of the following program?

```
class Hexy {

    public static void main (String[] args)    {
       Integer i = 42;
       String s = (i<40)?"life":(i>50)?"universe":"everything";
       System.out.println(s);
   }
}
```

A. null
B. life
C. universe
D. everything
E. Compilation fails
F. An exception is thrown at runtime.

Problem Statement:
Given:

```
1. class Example {
2.    public static void main(String[] args) {
3.       Short s = 15;
4.       Boolean b;
5.       // insert code here
6.    }
7.}
```

Which, inserted independently at line 5, will compile? (Choose all that apply.)

```
A.          b  =  (Number instanceof s);
B.          b  =  (s instanceof Short);
C.          b  =  s.instanceof (Short);
D.          b  =  (s.instanceof Number);
E.          b  =  s.instanceof (Object);
F.          b  =  (s instanceof String);
```

Problem Statement:
What is the output of the following program?

```
class TryIt {

   public static void main(String[] args) {
      Integer x = 0;
      Integer y = 0;
     for(Short z = 0; z < 5; z++)
         if ((++x > 2) || (++y > 2))
            x++;
      System.out.println(x + " " + y);
   }
}
```

Problem Statement:
What is the output of the following program?

```
class Titanic {

   public static void main(String[] args)    {
      Boolean b1 = true;
      Boolean b2 = false;
      Boolean b3 = true;
      if ((b1 & b2) | (b2 & b3) & b3)
         System.out.println("alpha ");
      if ((b1 = false) | (b1 & b3) | (b1 | b2))
         System.out.println("beta ");
   }
}
```

a) beta
b) alpha
c) alpha beta
d) Compilation fails.
e) No output is produced.
f) An exception is thrown at runtime.

Problem Statement:

Given the following program:

```
1. class Maybe {
2.    public static void main(String[] args) {
3.       boolean b1 = true;
4.       boolean b2 = false;
5.       System.out.println(!false ^ false);
6.       System.out.println(" " + (!b1 & (b2 = true)));
7.       System.out.println(" " + (b2 ^ b1));
8.    }
9. }
```

Which are true?
a) Line 5 produces true.
b) Line 5 produces false.
c) Line 6 produces true.
d) Line 6 produces false.
e) Line 7 produces true.
f) Line 7 produces false.

Problem Statement:
What is the output of the following program?

```
class Alien {
   String invade(short ships) {
      return " a few";
   }
}

class Defender {
   public static void main(String[] args) {
      System.out.println(new Alien().invade(7));
   }
}
```

A. many
B. a few
C. Compilation fails
D. The output is not predictable
E. An exception is thrown at runtime

Problem Statement:
What is the output of the following program?

```
class Fizz {
  int x = 5;
```

```
  public static void main(String[] args) {
    final Fizz f1 = new Fizz();
    Fizz f2 = new Fizz();
    Fizz f3 = FizzSwitch(f1, f2);
    System.out.println((f1 == f3) + " " + (f1.x == f3.x));
  }
  static Fizz FizzSwitch(Fizz x, Fizz y) {
    final Fizz z = x;
    z.x = 6;
    return z;
  }
}
```

A. true true
B. false true
C. true false
D. false false
E. Compilation fails.
F. An exception is thrown at runtime.

Java

1. J2SE - Java 2 Standard Edition - system/networking oriented project
2. J2EE - Java 2 Enterprise Edition - web oriented appl
3. J2ME - Java 2 Mobile/Micro Edition - mobile based appl

Sun Microsystem - Now it is taken by Oracle
late 1980 - Oak
early 1990 - Java - James Gosling, Chris Wirth, Patrick Naughton

JDK1.0 - Oak
JDK2.0 - Playground
JDK3.0 - Kestrel
4.0 - Mustang
5.0 - Tiger
7.0 - Dolphin
8.0 - Spyder

Fully OOP - Scala

C
1. Procedural lang
2. Header files
#include<stdio.h>
#include<math.h>

Java
1. Partially OOPS
2. No header files instead we use
Packages - folder which contains
collection of class files

```
import java.io.*;
import java.io.File;
import java.util.*;
import java.lang.*;
```

3. 32 keywords
no - sizeof, extern, register,

3. 51 Keywords
signed, unsigned

4. Primitive datatype,
user defined datatype

4. Only primitive datatype

5. #define A 40
Macros

5. No global declaration in java

6. Pointers

6. No Pointers

function - exists independently
method - exist inside class

C++
1. Partially OOP

Java
1. Partially OOP

a. object - real world entity, contain memory reference or address
b. class - template/blueprint
- class variables, methods and constructors

- access the class by creating an object

c. Encapsulation - wrapping of data into single unit(ie) class
- Data hiding/Information hiding
- class contain private variables and public methods

d. Abstraction - without knowing the background details of tbe class, we are using that class

e. Inheritance - accessing the properties of one class into another class
- code reusability

In C++, single, multiple, multi level, hybrid, hiearchial — In java, we dont have multiple inheritance instead we have Interface

f. Polymorphism - one class which takes many forms - 2 types

1. Static/Compile time polymorphism - using overloading

In C++, we have method, constructor and operator overloading — In Java, we cant overload operator

2. Dynamic/Runtime polymorphism - using overriding - only Method overriding

2. Constructor
- What is constructor?
  - Whenever we create an object for a class, memory will be allocated and allocation of memory is called constructor
- When constructor invoked?
  - Whenever we create an object for a class, the constructor will be invoked
- Why constructor?
  - used to initialize values to the variable at time of object creation
- By default all class will contain one default constructor (ie) constuctor without any parameter, and we can create our own constructor where ur class name and constructor name should be same, at that time the already present default constructor will be overlap
- Constructor can be created with or without parameters
- Constructor can have access specifiers
- Constructor should not have return type

```
class Box {
   Box() {  //default constructor

   }
   void Box(){   //method
   }
}
```

3. Destructor
- Deallocation of memory
- ~(tilde) operator

3. No destructor, instead we have
- Automatic garbage collection
- System.gc()/Runtime.gc() where it will internally call finalize() which is used to deallocate the resource and invoked only once in lifecycle of program

4. String is a datatype

4. String is a class/literal

Fundamental of Java

1. Identifiers
- name given for class, methods and variables
- should always starts with char, con contain digits, special char ($,_) allowed

Java Coding standard

1. class - starting letter of each word should be capital letter
   eg: ExampleProgram, SampleController
2. methods - camelcase, from 2nd word onwards each starting letter should be capital
   eg: getAge(), getNameOfEmployee()
3. variables - all lowercase
4. Constants - all uppercase

Formatted - ctrl+A - Ctrl+shift+F

2. Keywords
- 51 keywords
- goto and const are the keywords of Java, but when we use it gives compilation error
  goto - continue
  const - final
- 3 reserved words - true, false, null

```
boolean b1=true;
boolean b2=false;
```

object = null

3. Datatypes
- what type of value stored in a variable

datatype	byte	min	max
1. byte	1	-2^7	(2^7)-1 -128 to 127
2. short	2	-2^15	(2^15)-1
3. int	4	-2^31	(2^31)-1
4. long	8	-2^63	(2^63)-1
5. float	4	-	-
6. double	8	-	-
7. char	2(16 bit unsigned integer)	0	65535
8. boolean	1	false	true

4. Literal
- how we can define the value into a variable

a. Numeric literal - 3 ways to represent number in java
1. Decimal literal - base 10
```
int a=34;
int b=27363;
```
2. Octal literal - base 8 - always precedded with 0
```
int a=01;  //1
int b=02;  //2
   03,04,05,06,07
int c=010;  //8
    0*8^0=0
    1*8^1=8
    0*8^2=0
int d=011;  //9
    1*8^0=1
    1*8^1=8
    0*8^2=0
int e=012;  //10
    2*8^0=2
    1*8^1=8
    0*8^2=0
```
3. Hexadecimal literal - base 16 - always precedded with 0x or 0X
```
int a=0x16;  //22
    6*16^0=6
    1*16^1=16
```

b. short literal
```
short s1=10;  //correct
short s2=10s;  //error
short s3=10S;  //error
```

c. Long literal
```
long l1=10;  //correct
long l2=10l;  //correct
long l3=10L;  //correct
```

By default in Java, all +ve or -ve number it is considerd as int datatype

d. Float literal
```
float f1=3.14;  //error
float f2=3.14f;  //correct
float f3=3.14F;  //correct
```

e. Double literal
```
double d1=3.14; //correct
double d2=3.14d; //correct
double d3=3.14D; //correct
```

By default in Java, all decimal number it is considerd as double datatype

f. Boolean literal
```
boolean b1=true;  //correct
boolean b2=false; //correct
```

```
boolean b3=True;  //error
```

g. char literal - single char single quotes

```
char c1='a';  //correct
char c2="a";  //error
char c3='ab';  //error
char c4="ab"; //error
char c5=23;   //correct (prints ascii value for that 23)
char c6=12733; //correct
char c7=-45;  //error
char c8=(char)70000; //correct
```

In Java, char can also be represented as unicode representation

```
char c='\u0000';
```

h. String literal - sequence of char with double quotes

```
String s1="hello";
```

i. null literal - can be declared only object

```
String s1=null;
Box b=null;
```

5. Variables
- identifier used to store the value - 2 types

```
int a=10;
```

1. Instance/class variable
- Any variable that is declared inside the class and outside the method

```
class A {
   int a;  //Instance variable
   void add(){
   }
}
```

- No need to initialize instance variable, it will take default value depending on datatype

int,byte,short,long - 0
float,double - 0.0
boolean - false
char - '\u0000'
any object - null

2. Local variable
- Any variable that is declared inside the method and complusorly should be initialized otherwise compilation error

```
class A {
   int a;  //Instance variable
   void add(){
     int b=0;  //Local variable
   }
}
```

6. Access specifiers/Access modifiers - 4 types
1. public - it can be accessed anywhere
2. private - it can be accessed only within the class in which it is declared
3. protected - it can be accessible within the class as well as in its inherited class
4. default - if we not specify public or private or protected, then by default it is considered as default specifier, it is default access specifier in Java

Visibility	default	private	protected	public
1. same pkg same class	yes	yes	yes	yes
2. same pkg different class	yes	no	yes	yes
3. Different pkg subclass	no	no	yes	yes
4. Different pkg non subclass	no	no	no	yes

7. Type conversion
- converting from one datatype to another datatypes

1. Implicit conversion - automatic conversion from lower to higher datatype
int a=10;
double d=a; //implicit
2. Explicit conversion - conversion from higher to lower datatype

```
int i=128;
byte b=(byte)i;  //explicit    -128 to 127
sop(b);  //-128

int i=129;
byte b=(byte)i;  //explicit    -128 to 127
sop(b);  //-127
```

8. Operators
1. Arthimetic operators +,-,*,/
2. Modulus operator %
3. Relational operator >,<,>=,<=
4. Assignment operator =
5. Conditional Assignment operator +=,-=,*=,/=
a+=2; a=a+2;
6. Equality and Inequality operator ==,!=
7. Ternary operator ?:
z=a>b?a:b;
8. Increment and Decrement operator ++,--
9. Bitwise operator - always works on truth table
& - Bitwise AND
| - Bitwise OR
^ - Bitwise XOR
~ - 1's complement
>> - Right shift - n/2^s
<< - Left shift - n*2^s

a=5, b=6;

a	b	a&b=4	a|b=7	a^b=3	~a
0	0	0	0	0	1
1	1	1	1	0	0
0	1	0	1	1	1
1	0	0	1	1	0

a=8, b=2
a>>b=8/2^2=8/4=2
a<<b=8*2^2=8*4=32

10. Boolean logical operator a=true, b=false, a&b=?
& - Logical AND
| - Logical OR
! - Logical NOT

11. Shortcircuit logical operator - used to check some condition
&& ||

```
if(a>10 && b==1){

}
```

12. new operator - used to create an object for the class - 3 ways

```
class Box {
  int a=10;
  void show(){
  }

  Box(String s){  //parameterized constructor

  }
}
```

1. Box b=new Box();
- We are creating an object called "b", and we are allocating the memory using new operator and stored inside the object and invokes ur default constructor

Box b1=new Box("hello");

2. Box b; //object declaration which contains null reference
b=new Box(); //creating object, allocate memory, invoke constructor
b.show();

We create object to access method and variable

3. new Box(); //allocate and invoke constrcutor
new Box().show(); //allocate, invoke constructor and call show()

13. instanceof operator
- used to check whether an object is an instance of that class

```
Box b=new Box();
if(b instanceof Box){
  //logic
}
```