1. Finalise method is called at the time of garbage collection. System.gc() internally calls it.
2. == and equals: == compares the primitives or address location of the object while equals compares the actual values of object.
3. Static methods can be called directly using the classname without creating instance of the class. Can be used for Eager loading
4. Detect and debug exceptions using try and catch block, and adding breakpoints at the particular line of code.
5. Using hashing, Encrypt it using some hardcoded text with Base64encoder.
6. Convert Object String into some other text format
7. Block 1 is static, will be eager loaded with class. Value of a is always 5 while value of b will be 0 at the time of loading.
8. Floating point
9. B and D
10. D
11. F
12. C
13. C
14. Problem solving:

forename

InStr('JONES,NICK', ',',1)//6

SubStr ('JONES,NICK’,6) ->NICK

Surname

InStr('JONES,NICK', ',',-1)//5

SubStr ('JONES,NICK’,5,1) ->JONES

public String forename(String fullname){

int position= InStr(fullname, ',',1);

return SubStr (fullname,position);

}

public String surname(String fullname){

int position= InStr(fullname, ',',-1);

return SubStr (fullname,position,1);

}

1. SQL:
2. Groupby is used to group a set of entities based on one or more parameters.
3. Orderby is used to sort the table in ascending or descending order
4. Having can be used after groupby to add certain logical conditions with aggregate functions
5. Minus is used as a mathematical operator returns unique records.
6. Inner join returns the matched values from both tables.

Left outer join when the reference table is maintained at the left side to match other tables.Right Outer join is vice versa

1. Deadlock is a situation where we wait for an entity which is already occupied by another.Using transaction management, we can avoid deadlocks.
2. Exclusive
3. Best practice of query optimization is adding indexes.
4. We can create own indexes for sorting related stuff.
5. No
6. Programming:

public Class Node{

Node next;

Node prev;

int data;

Node(int data){

}

public class MyDoublyLinkedList{

Node head=null;

Node tail=null;

public void addObject(int data, int position){

Node newNode = new Node(data);

int currentposition=1;

If(currentposition<position){

if(head == null) {

head = tail = newNode;

head.previous = null;

tail.next = null;

}

else {

head.next = newNode;

newNode.prev=head;

newNode.next=null;

}

}

public static void main(String[] args) {

MyDoublyLinkedList myList = new MyDoublyLinkedList ();

myList.addObject (A);

1. XML/HTML

a) XML Extensible markup language: It is file used over protocols to display the content in readable format

b) XSD is schema definition, kind of a template used for formatting document.

c) ordinality

d) <table><th><td><tr></tr>><tr></tr></td>><td><tr></tr>><tr></tr></td></th></table>

e) cascading style sheet- styling sheet to define the view of a page.

7. Analysis

a)   
1. Understanding the requirement thoroughly, ask questions in case of doubts.

2. Have an initial level planning discussion with stakeholders briefing your understanding with the help of BRD and HLD documentation with below points:

* UML diagram
* sequence diagram using the models in above.
* flow diagram to understand the sequence of events.
* Analyze the impact and challenges.

3. Requirement discussion with team. Planning and addressing the usecases.

4. Create JIRA stories.

5. Add estimates, then Coding, unit testing, code reviews, defect fixing and so on.

b) Usecases are some tasks, actions or processes which can be implemented in our application as a new feature or enhancement. Effort can be measured in terms of time, complexity and other factors.

c) A simple usecase is developing REST API to create, get and modify customers.

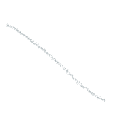
1. identify the actors (why, what, how, when and where)

2. Create Business and high level document for the requirement

Flow:

3. Share requirement to the team

4. Create JIRA story with complexity

Customer fields- Id, firstName, lastName, Age

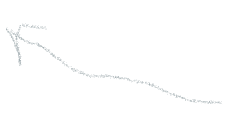
|  |
| --- |
| **Customer** |
| id |
| firstName |
| lastName |
| age |
| //getters setters |

|  |
| --- |
| **Customer Controller** |
|  |
| +Createcustomer() : String |
| +getcustomer(): List<Customer> |
| +updateCustomer():  Customer |

|  |
| --- |
| **Customer entity** |
| id |
| firstName |
| lastName |
| age |
| //getters setters |

|  |
| --- |
| **Customer Service** |
|  |
| +updateCustomer:Customer |
| +saveCustomer:String |
| +findCustomers:List<Customer> |







CustomerRepository



Customer POJO class- Customer TO class - Customer entity class

Controller layer -Service layer- DAO layer- Database

@RestController CustomerController.java

@Getmapping

public List<Customer> getCustomers(){

}

@postmapping

public String createCustomers(){

}

@putmapping

public Customer modifyCustomers(){

}

@Service - CustomerService

public List<Customer> getCustomersService(){

customerRepository.findallcustomers();

}

public String createCustomersService(){

customerRepository.save()

}

public Customer modifyCustomerService(){

customerRepository.update();}

@Repository - CustomerRepository implements JPArepository