**Problem 1:**

|  |
| --- |
| **public** **class** Problem1 {  **public** **static** **void** main(String[] args) {  Parent p = **new** Child("Parent");  System.***out***.println(p.getName());  }  }  **class** Parent {  **protected** String name;  **public** Parent(String name) {  **this**.name = name;  }  **protected** String getName() {  **return** **this**.name;  }  }  **class** Child **extends** Parent {  **public** Child(String name) {  **super**(name);  **this**.name = name;  }  @Override  **protected** Object getName() {  **return** **this**.name;  }  } |

**Problem 2:**

|  |
| --- |
| **public** **class** Problem2 {  **interface** CustomPredicate<T> {  **boolean** eval(T t);  }  **public** **static** **void** main(String[] args) {  List<String> sampleData = Arrays.*asList*("spring", "node", "mkyong");  CustomPredicate<String> filterStingsWithNPrefix = data -> data.startsWith("N");  sampleData.stream().filter(filterStingsWithNPrefix).forEach(System.***out***::println);  }  } |

**Problem 3:**

|  |
| --- |
| **public** **class** Problem3 {  **public** **static** **void** main(String[] args) {  Parent1 p1 = **new** Child1();  p1.method1();  p1.method1("Overloded method");  }  }  **class** Parent1 {  **protected** **void** method1() {  System.***out***.println("Method one Parent");  }  **protected** **void** method1(String arg) {  System.***out***.println(arg + " Parent");  }  }  **class** Child1 **extends** Parent1 {  @Override  **protected** **void** method1() **throws** CustomRuntimeException {  System.***out***.println("Method one Child");  }  @Override  **protected** **void** method1(String arg) **throws** CustomException {  System.***out***.println(arg + " Child");  }  }  **class** CustomException **extends** Exception {  **protected** CustomException() {  **super**();  }  }  **class** CustomRuntimeException **extends** RuntimeException {  **protected** CustomRuntimeException() {  **super**();  }  } |

**Problem 4:**

|  |
| --- |
| **public** **class** Problem4 {  **public** **static** **void** main(String[] args) {  Parent3 p3 = **new** Child3();  p3.method1("Child");  }  }  **class** Parent3 {  **protected** **void** method1(String... args) {  System.***out***.println("Parent method");  }  }  **class** Child3 **extends** Parent3 {  **protected** **void** method1(String arg) {  System.***out***.println("Child method");  }  } |

**Problem 5:**

|  |
| --- |
| **public** **class** Problem5 {  **static** {  System.***out***.println("First Static block");  **new** Problem5().m1();  }  **static** **int** *i* = 10;  **public** **static** **void** main(String[] args) {  System.***out***.println("Main methodd");  System.***out***.println("j: " + *j*);  }  **private** **void** m1() {  System.***out***.println("i: " + *i*);  }  **static** {  System.***out***.println("Second static block");  }  **static** **int** *j* = 10;  } |

**Problem 6:**

|  |
| --- |
| **//Problem 5 class defined above**  **public** **class** Problem6 **extends** Problem5 {  **static** **int** *x* = 10;  **static** {  **new** Problem6().m2();  System.***out***.println("Derived class first static block");  }  **public** **static** **void** main(String[] args) {  **new** Problem6().m2();  System.***out***.println("Derived class main method");  }  **private** **void** m2() {  System.***out***.println("y: " + *y*);  }  **static** {  System.***out***.println("Derived second static block");  }  **static** **int** *y* = 200;  } |

**Problem 7:**

|  |
| --- |
| **public** **class** Problem7 {  **public** **static** **void** main(String[] args) {  String date = LocalDateTime.*parse*("2014-05-04").format(DateTimeFormatter.***ISO\_DATE\_TIME***);  System.***out***.println(date);  }  } |

**Problem 8:**

|  |
| --- |
| **public** **class** Problem8 {  **public** **static** **void** main(String[] args) {  Predicate<Student> p1 = s -> s.stuName.startsWith("A");  Predicate<Student> p2 = s -> s.stuAge < 40;  Predicate<Student> p3 = s -> s.stuAge < 40 && s.stuName.startsWith("P");  System.***out***.println(Student.*getStudents*().stream().anyMatch(p1));  System.***out***.println(Student.*getStudents*().stream().anyMatch(p2));  System.***out***.println(Student.*getStudents*().stream().anyMatch(p3));  }  }  **class** Student {  **int** stuId;  **int** stuAge;  String stuName;  Student(**int** id, **int** age, String name) {  **this**.stuId = id;  **this**.stuAge = age;  **this**.stuName = name;  }  **public** **static** List<Student> getStudents() {  List<Student> list = **new** ArrayList<>();  list.add(**new** Student(11, 28, "Lucy"));  list.add(**new** Student(28, 27, "Tim"));  list.add(**new** Student(32, 30, "Daniel"));  list.add(**new** Student(49, 27, "Steve"));  **return** list;  }  } |

**Problem 9:**

|  |
| --- |
| **public** **class** Problem9 {  **public** **static** **void** main(String[] args) {  Optional<String> GOT = Optional.*of*("Game of Thrones");  Optional<String> nothing = Optional.*empty*();  **if** (GOT.isPresent()) {  System.***out***.println("Watching Game of Thrones");  } **else** {  System.***out***.println("I am getting Bored");  }  GOT.ifPresent(s -> System.***out***.println("Watching GOT is fun!"));  nothing.ifPresent(s -> System.***out***.println("I prefer getting bored"));  }  } |

**Problem 10:**

|  |
| --- |
| **public** **class** Problem10 **implements** Runnable {  **public** **static** Problem10 *obj*;  **private** **int** data;  **public** Problem10() {  data = 10;  }  @Override  **public** **void** run() {  *obj* = **new** Problem10();  *obj*.wait();  *obj*.data += 20;  System.***out***.println(*obj*.data);  }  **public** **static** **void** main(String[] args) **throws** InterruptedException {  Thread thread1 = **new** Thread(**new** Problem10());  Thread thread2 = **new** Thread(**new** myThread());  thread1.start();  thread2.start();  System.***out***.printf(" GFG - ");  }  }  **class** myThread **implements** Runnable {  @Override  **public** **void** run() {  Problem10.*obj*.notify();  }  } |

**Problem 11:** What is the scope of stateful bean?

**A** - session

**B** - global-session

**C** - prototype

**D** – request

**Problem 12:** Are Singleton beans are thread safe in Spring Framework?

**A** – Yes

**B** – No

**Problem 13:** What are the different types of events of Listeners?

**A** - ContextClosedEvent.

**B** - RequestHandledEvent.

**C** - ContextRefreshedEvent.

**D** - All of the above

**E** - A & B

.

**Problem 14:** Name all Auto-wiring techniques in Spring and name the default.

**Problem 15:** Is REST?

**A** - Scalable.

**B** - Interoperable.

**C** – Both A & B.

**D** – None of the above

**Problem 16:** Name the HTTP methods attached with CRUD operations?

**A** - Create.

**B** - Read.

**C** - Update.

**D** – Delete

**Problem 17:** Different type of indexes in SQL?

**A** – Unique Index.

**B** – Clustered Index.

**C** – Non clustered index.

**D** – A & B

**E** – All of the above.

**Answer:**

1. Compilation Failure
2. Compilation Failure
3. Compilation Failure
4. Parent method

|  |
| --- |
| First Static block  i: 0  Second static block  Main method  j: 10 |



|  |
| --- |
| First Static block  i: 0  Second static block  y: 0  Derived class first static block  Derived second static block  y: 200  Derived class main method |

1. java.time.format.DateTimeParseException

|  |
| --- |
| false  true  false |



|  |
| --- |
| Watching Game of Thrones  Watching GOT is fun! |

1. Compilation Failure
2. C.
3. B.
4. D.
5. Autowire by – name, type, constructor and autodetect, Default is Autowiring by type
6. D.
7. Create – POST, Read – GET, Update – PUT, Delete – DELETE.
8. E.