Q1) What is not true about the scanner?

1. It can be used to read from the file and console
2. Used for pattern matching
3. It is from java.util package
4. It parses input into tokens using delimiter to identify token boundary
5. Default delimiter is a semi-colon

Q2) What would be the output of the following program?

public class ParseString

{

public static void main(String[] args)

{

Scanner scanner = new Scanner("1, 2, 3, 4, 5, 6,7,8").useDelimiter(", ");

while (scanner.hasNextInt()) {

int num = scanner.nextInt();

if (num % 2 == 0)

System.out.println(num);

}

}

}

1. 2 4 6 8
2. 2 4
3. 1 2 3 4 5 6 7 8
4. Compile error
5. Runtime error

Q3) What would be the output:

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

StringBuffer buffer = **new** StringBuffer("1 min ");

buffer.append("Hi");

buffer.insert(1, " Wait");

buffer.reverse();

System.***out***.println(buffer);

}

}

1. Compile error
2. iH nim tiaW 1
3. Hi min wait 1
4. Runtime error

Q4) Which of the following are true:

1. Duration is used to measure diff in dates in terms of days, months, years
2. Period is used to measure time including nanoseconds precision
3. Period is used to measure diff in dates in terms of days, months, years
4. Duration is used to measure diff in dates in terms of hours, minutes, seconds
5. Duration is used to represent machine timestamp

Q5) What would be the output of this snippet?

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

LocalDateTime t= LocalDateTime.*now*();

t.plusDays(4);

System.***out***.println(t.getMonth());

}

}

1. Compile error
2. Runtime error
3. 09
4. SEPTEMBER

Q6) What would be the output?

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

LocalDate t= LocalDate.*now*();

DateTimeFormatter formatter= DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***SHORT***);

System.***out***.println(t.format(formatter));

}

}

1. Compile error
2. 25/9/17
3. Monday, 25 September, 2017
4. 25 Sep, 2017
5. None of these

Q7)

What would be the output?

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

B b= **new** B();

}

}

**class** A{

**int** x= 5;

A(){

**super**(); //1

}

}

**class** B **extends** A{

**int** x= 4;

B(){

**super**(); //2

System.***out***.println(**super**.x); //3

System.***out***.println(**this**.x); //4

}

}

1. 5 4
2. 4 5
3. Error at line 1
4. Error at line 3,4

Q8) Choose the correct option

1. Abstract class is one without any child classes
2. Abstract class is any parent class with more than 1 child class
3. Abstract class cannot be instantiated
4. Abstract class is another name for base class

Q9)Fill in the blank. Class Valentine, Holiday and Birthday are child classes of Card class

\_\_\_\_\_\_\_\_\_ card;

Card= new Valentine();

Card.greeting();

Card= new Holiday();

Card.greeting();

Card= new Birthday();

Card.greeting();

1. Valentine
2. Holiday
3. Birthday
4. Card

Q10)Which of these are a valid declaration in an interface?

1. public static short stop= 23;
2. protected short stop=23;
3. public final short stop=23;
4. short stop =34;

Q11) Which assignments are legal?

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

B b= **new** B();

C c= **new** C();

D d= **new** D();

}

}

**interface** A{}

**class** B{}

**class** C **extends** B **implements** A{}

**class** D **implements** A{}

1. A obj = d;
2. B obj = c;
3. C obj = d;
4. D obj = (D) c;

Q12) What would be the output of this snippet?

**public** **class** Demo2

{

**public** **static** **void** main(String[] args)

{

C c= **new** C();

*m1*(c);

}

**static** **void** m1(A a){

System.***out***.println("1");

}

**static** **void** m1(B a){

System.***out***.println("2");

}

**static** **void** m1(Object o){

System.***out***.println("3");

}

}

**class** A{}

**class** B **extends** A{}

**class** C **extends** B{}

1. Compile error
2. 2
3. 1
4. 3
5. Runtime error

Q 13)

**public** **class** Demo2

{

**int** m1(**int** i, **int** d){

**return** i+d;

}

**static** **int** m1(**int** i, **double** d){

**return** (**int**)(i+d);

}

**double** m1(**double** i, **int** d){

**return** i+ d;

}

**static** **double** m1(**double** i, **double** d){

**return** i+d;

}

}

Is method m1 overloaded successfully?

1. Compile Error
2. Runtime error
3. Overloaded
4. Not overloaded, but will work

Q14) What would be the output of this snippet?

**public** **class** Demo2 {

**public** **static** **void** main(String args[]){

**new** Y().m1(100);

}

}

**class** X1{

**void** m1(**int** a){

System.***out***.println("1");

}

**void** m1(**double** a){

System.***out***.println("2");

}

}

**class** Y **extends** X1{

@Override

**void** m1(**double** d){

System.***out***.println("3");

}

}

1. 2
2. 3
3. Compile error
4. Runtime error
5. 1

Q15) Which of these is not an overridden method: produces error?

1. **class** X1{

Object m1(**double** a){

**return** 1;

}

}

**class** Y **extends** X1{

@Override

Object m1(**double** d){

**return** 2;

}

}

1. **class** X1{

Object m1(**double** a){

**return** 1;

}

}

**class** Y **extends** X1{

@Override

Integer m1(**double** a){

**return** 3;

}

}

1. **class** X1{

**int** m1(**double** a){

**return** 1;

}

}

**class** Y **extends** X1{

@Override

Integer m1(**double** a){

**return** 3;

}

}

1. **class** X1{

**int** m1(**double** a){

**return** 1;

}

}

**class** Y **extends** X1{

@Override

**int** m1(**double** a, **double** d){

**return** 3;

}

}

Q16) Which of these snippet will cause compile error?

1. **public** **class** Demo2 {

**final** **int** x;

**static** **final** **int** ***y***;

{

x= 0;

}

**static**{

***y*** = 4;

}

}

1. **public** **class** Demo2 {

**final** **int** x;

**static** **final** **int** ***y***;

Demo2(){

x= 0;

}

**static**{

***y*** = 4;

}

}

1. **public** **class** Demo2 {

**final** **int** x= 5;

**public** **void** m1(){

**final** **int** x= 4;

}

}

1. **public** **class** Demo2 {

**final** **int** x= 5;

**public** **void** m1(){

**int** x= 4;

}

}

1. **public** **class** Demo2 {

**public** **void** m1(){

**final** **int** x;

}

}

1. None of the above

Q17) Which of these are incorrect about the final modifier?

1. Only the final modifier can be used with the local variable. No other modifier can be used with local variables
2. Final methods cannot be overridden or overloaded
3. Final class cannot be extended
4. We cannot use final modifier with the interface or abstract class
5. Final variables cannot be reassigned a value

Q18) Which of these cannot be declared abstract

1. Variables
2. Constructors
3. Static methods
4. Class
5. Interface

Q19) Which of these is false?

1. Child Interface can use the default method of parent interface.
2. Child Interface can re-declare the default method without default keyword to

make it abstract.

1. Child Interface can override the default method by keeping the same signature

as of parent interface.

1. Default methods like abstract methods do not have a method body.
2. When a child class implements 2 interfaces with same default method, it needs to provides it’s own implementation for that method.

Q20) Which of these are true about interfaces?

1. Default methods can be public, private, protected or default scoped.
2. Static methods can be public, private, protected or default scoped.
3. Interface can extends any number of interfaces
4. A class can extends only 1 class but can implement any number of interfaces

Q21) Which of these statements are correct?

1. Polymorphism does not work with static methods
2. All variables in interfaces are public , static, final
3. Regex classes are from java.util.regex package
4. Regex classes are from java.lang.regex package
5. Regex can be used to search the text but cannot be used to change the value

Q22) Which of these are correct?

1. Compile method of Pattern class is used to create an object of a pattern class
2. matcher method of pattern class is used to create object of matcher class
3. matches(), find(), start(), group(), end() are methods of pattern class
4. Pattern [a-z && [def]] means either d or e or f
5. Pattern [a-z && [^m-p]] means starting with anything between m and p
6. Pattern (^[a-z]{4}$) means exactly 4 characters in lowercase

----Exception Handling----

Q23) Which of these is true about exception handling?

1. When an exception occurs, the exception object is created and handed over to JVM
2. A try block must have a catch block else compile error
3. Exceptions provide the mean to separate details of what to do when error occurs
4. The parent of all exceptions and error classes is Exception class
5. All exception classes are in java.util package
6. Stack overflow is an example of unchecked exception

Q24) Which of these is true about checked exceptions?

1. They should be handled using try/catch or using the throw keyword
2. They represent non-fatal errors
3. They are checked at the runtime
4. IOException, FileNotFoundException, SQLException, ClassNotFoundException are some examples of checked exceptions

Q25) Which of these is true about unchecked exceptions?

1. These are considered fatal for the program
2. They are checked at compile time
3. SQLException is a checked exception
4. To create a unchecked exception class, you must extend RuntimeException class

Q26)Which of these is true about exception handling?

1. Each try block must have a catch block
2. You can have nested try blocks in the code
3. You can throw an error using throw keyword in either try block or a catch block or a finally block
4. Throws is used to throw the new exception
5. We can use either catch block or finally block to catch the exceptions
6. Try with resource block can exist without catch or finally
7. Try with resource block is used to close the resources that implement java.lang.AutoCloaseable interface

Q27) Exception class is in which package?

1. java.io
2. java.lang
3. java.util
4. java.file

Q28) Which of these code snippets throw error?

1. **public** **class** Demo2 {

**public** **static** **void** main(String args[]){

**try**{

**throw** **new** Derived();

}**catch**(Base | Derived b){

System.***out***.println(b);

}

}

}

**class** Base **extends** Exception{}

**class** Derived **extends** Base{}

1. **public** **class** Demo2 {

**public** **static** **void** main(String args[]){

**try**{

**throw** **new** Derived();

}**catch**( Derived | Base b){

System.***out***.println(b);

}

}

}

**class** Base **extends** Exception{}

**class** Derived **extends** Base{}

1. **public** **class** Demo2 {

**public** **static** **void** main(String args[]){

**try**{

**throw** **new** Derived();

}**catch**( Derived b){

System.***out***.println(b);

}**catch**( Base c){

System.***out***.println(c);

}

}

}

**class** Base **extends** Exception{}

**class** Derived **extends** Base{}

1. **public** **class** Demo2 {

**public** **static** **void** main(String args[]){

**try**{

**throw** **new** Derived();

}**catch**( Base c){

System.***out***.println(c);

}**catch**( Derived b){

System.***out***.println(b);

}

}

}

**class** Base **extends** Exception{}

**class** Derived **extends** Base{}

Q29) Match the following

