1. Che cosa sono i nomi e l'ambiente (environment)? I nomi sono identificatori che vengono usati per fare riferimento a variabili, funzioni e altre entità all'interno di un programma. L'ambiente, o contesto di esecuzione, è l'insieme di nomi e dei loro valori associati che sono attualmente disponibili in un certo punto del programma. L'ambiente può essere influenzato dalla presenza di strutture come funzioni annidate o variabili globali.
2. Che cosa sono le regole di visibilità? Le regole di visibilità descrivono le condizioni sotto le quali un nome è accessibile all'interno di un programma. In genere, i nomi definiti in un blocco di codice sono visibili solo all'interno di quel blocco e dei suoi sotto-blocchi, a meno che non vengano dichiarati come variabili globali. Le funzioni annidate possono accedere alle variabili definite nella loro funzione genitore, ma non viceversa.
3. Si spieghi la gestione della memoria dinamica tramite stack e record di attivazione. Il programma può allocare memoria dinamica tramite l'utilizzo dello stack e dei record di attivazione. Il record di attivazione rappresenta l'attuale attivazione di una funzione o di una procedura e viene creato ogni volta che viene chiamata una funzione. Esso contiene informazioni come gli argomenti passati alla funzione, i valori delle variabili locali e il puntatore di ritorno. Il record di attivazione viene inserito nello stack, che è una regione di memoria in cui gli oggetti sono allocati e deallocati in ordine LIFO (Last In, First Out). Quando una funzione termina, il suo record di attivazione viene rimosso dallo stack e la memoria associata viene liberata.
4. Si illustrino le operazioni che sono eseguite quando viene eseguita una procedura. Quando viene eseguita una procedura, il controllo del programma viene trasferito alla procedura stessa. Vengono creati uno o più record di attivazione nello stack per la procedura e i suoi eventuali sotto-procedure o blocchi di codice. Vengono poi eseguite le istruzioni all'interno della procedura, che possono includere l'accesso a variabili locali o argomenti passati alla procedura. Quando la procedura termina, il valore di ritorno viene restituito al chiamante e il record di attivazione viene rimosso dallo stack.
5. Si spieghino le possibili strategie per la gestione della memoria dinamica tramite heap. Si illustrino la differenza tra linguaggi imperativi e dichiarativi. La memoria dinamica può anche essere gestita tramite l'heap, una regione di memoria in cui gli oggetti sono allocati e deallocati in modo più flessibile rispetto allo stack. Le strategie per la gestione della memoria tramite l'heap includono l'allocazione e la liberazione esplicita della memoria, oppure l'utilizzo di un meccanismo di garbage collection. I linguaggi imperativi come C o C++ richiedono che il programmatore

gestisca esplicitamente l'allocazione e la liberazione della memoria dinamica tramite le funzioni come `malloc` e `free`. Ciò richiede una maggiore attenzione alla gestione della memoria, ma fornisce anche maggiore controllo sul momento in cui la memoria viene allocata e liberata.

I linguaggi dichiarativi come Python o JavaScript, d'altra parte, spesso utilizzano un meccanismo di garbage collection per gestire la memoria dinamica. Il garbage collector è un processo che individua e rimuove gli oggetti non più utilizzati nella memoria dell'applicazione, liberando così la memoria inutilizzata. Ciò può semplificare la gestione della memoria per il programmatore, ma può anche causare un overhead di esecuzione e talvolta problemi di prestazioni.

6. Si parli del goto. Il goto è un'istruzione che consente di saltare in modo non strutturato a una posizione diversa nel codice. L'uso del goto può rendere il codice difficile da leggere e da comprendere, poiché può rendere difficile determinare l'ordine in cui le istruzioni vengono eseguite. Inoltre, il goto può rendere il codice più difficile da mantenere e modificare. Per questo motivo, molti linguaggi di programmazione moderni evitano di includere l'istruzione goto.
7. Si parli dei comandi condizionali. I comandi condizionali consentono di eseguire un blocco di codice solo se una certa condizione è verificata. In molti linguaggi di programmazione, la sintassi dei comandi condizionali prevede l'utilizzo di un'istruzione if, seguita dalla condizione da verificare e dal blocco di codice da eseguire se la condizione è vera. In alcuni casi, può essere utilizzato anche un'istruzione else per specificare un blocco di codice da eseguire se la condizione non è vera.
8. Si parli dei comandi iterativi. I comandi iterativi consentono di eseguire un blocco di codice ripetutamente finché una certa condizione è verificata. In molti linguaggi di programmazione, la sintassi dei comandi iterativi prevede l'utilizzo di un'istruzione while o for, seguita dalla condizione da verificare e dal blocco di codice da eseguire finché la condizione è vera. In alcuni casi, può essere utilizzato anche un'istruzione do-while per eseguire il blocco di codice almeno una volta prima di verificare la condizione.
9. Che cos'è la programmazione strutturata? La programmazione strutturata è un paradigma di programmazione che prevede l'utilizzo di costrutti di controllo strutturati come comandi condizionali e iterativi per scrivere codice in modo ordinato e organizzato. L'obiettivo della programmazione strutturata è di creare codice leggibile e facilmente comprensibile, evitando le pratiche di programmazione non strutturate come l'uso del goto.
10. Che cos'è la tail recursion? La tail recursion è un tipo di ricorsione in cui la chiamata ricorsiva è l'ultima oper

azione eseguita all'interno della funzione. In altre parole, la funzione ricorsiva restituisce il risultato della chiamata ricorsiva, senza eseguire altre operazioni. La tail recursion è importante perché consente di ottimizzare il codice, eliminando la necessità di mantenere una pila di chiamate ricorsive. In alcuni linguaggi di programmazione, il compilatore può riconoscere la tail recursion e ottimizzare automaticamente il codice per migliorare le prestazioni.

11. Che differenza c'è tra ricorsione e iterazione? La ricorsione e l'iterazione sono due tecniche di programmazione utilizzate per eseguire un blocco di codice ripetutamente. Nella ricorsione, una funzione richiama se stessa ripetutamente fino a raggiungere una condizione di uscita. Nell'iterazione, un blocco di codice viene eseguito ripetutamente finché una certa condizione è verificata.

La differenza principale tra ricorsione e iterazione è il modo in cui viene gestita la memoria. Nella ricorsione, vengono create nuove istanze della funzione sulla pila di chiamate ricorsive, che può portare a problemi di memoria se non viene gestita correttamente. Nell'iterazione, invece, viene eseguito un blocco di codice ripetutamente, senza creare nuove istanze di funzioni.

12. Quando un linguaggio ha first-class functions? Un linguaggio ha le funzioni di prima classe (o first-class functions) quando le funzioni possono essere trattate come qualsiasi altro valore del linguaggio. Ciò significa che le funzioni possono essere assegnate a variabili, passate come argomenti ad altre funzioni e restituite come risultato di una funzione.
13. Che cosa fa il garbage collector e perché è utile? Il garbage collector è un processo che individua e rimuove gli oggetti non più utilizzati nella memoria dell'applicazione, liberando così la memoria inutilizzata. Il garbage collector è utile perché semplifica la gestione della memoria per il programmatore, consentendo di evitare la necessità di gestire esplicitamente l'allocazione e la liberazione della memoria dinamica.
14. Si illustrino le due tecniche principali per la garbage collection. Le due tecniche principali per la garbage collection sono la mark-and-sweep e la reference counting. Nella mark-and-sweep, il garbage collector identifica gli oggetti inutilizzati esplorando il grafo degli oggetti e marcando quelli che sono ancora in uso. Successivamente, vengono rimossi tutti gli oggetti non marcati. Nella reference counting, il garbage collector tiene traccia del numero di riferimenti ad ogni oggetto e rimuove gli oggetti con un conteggio di riferimenti pari a zero.
15. Come sono implementati gli oggetti? Gli oggetti sono implementati come strutture di dati che contengono una serie di attributi o proprietà e metodi che consentono di manipolare quegli attributi o proprietà. In molti linguaggi di programmazione orientati agli oggetti, gli oggetti sono implementati come istanze di classi, che definiscono la struttura e il comportamento degli oggetti.
16. Come si svolge il dynamic method lookup? Il dynamic method lookup è il processo mediante il quale viene determinato il metodo da invocare su un oggetto in fase di esecuzione. Nella programmazione orientata agli oggetti, l'oggetto su cui viene eseguita un'operazione può essere di una classe diversa da quella dichiarata per la variabile che lo contiene. In questo caso, il dynamic method lookup determina quale metodo deve essere chiamato sulla base della classe effettiva dell'oggetto.

Il dynamic method lookup avviene in tempo di esecuzione, a differenza del metodo statico, che avviene in fase di compilazione. Quando viene invocato un metodo su un oggetto, il runtime del linguaggio esamina la classe effettiva dell'oggetto e cerca il metodo appropriato in quella classe. Se il metodo non viene trovato, viene cercato nella superclasse della classe effettiva, e così via, fino a quando viene trovato il metodo corretto.

17. Come sono rappresentate le classi? Le classi sono rappresentate in diversi modi a seconda del linguaggio di programmazione. In molti linguaggi di programmazione orientati agli oggetti, le classi sono rappresentate come strutture di dati che definiscono la struttura e il comportamento degli oggetti. Le classi contengono un insieme di attributi o proprietà e un insieme di metodi che consentono di manipolare quegli attributi o proprietà.

Nella maggior parte dei linguaggi di programmazione, le classi sono dichiarate in modo esplicito nel codice sorgente, e il compilatore genera il codice per creare gli oggetti delle classi durante l'esecuzione del programma.

18. Come si invocano i metodi nel caso di ereditarietà singola? Nel caso di ereditarietà singola, i metodi di una classe base possono essere invocati direttamente sui oggetti delle sottoclassi. Ad esempio, se una classe B eredita da una classe A, un oggetto di tipo B può chiamare direttamente i metodi ereditati dalla classe A.

    Se una classe B ridefinisce un metodo ereditato dalla classe A, l'invocazione del metodo su un oggetto di tipo B chiamerà la versione del metodo definita nella classe B, anziché la versione nella classe A.

19. Che cos'è la programmazione basata su eventi in JavaFX? La programmazione basata su eventi in JavaFX è un paradigma di programmazione utilizzato per creare interfacce utente reattive e interattive. In JavaFX, gli eventi vengono generati quando l'utente interagisce con l'interfaccia utente, ad esempio facendo clic su un pulsante o digitando un tasto sulla tastiera. La programmazione basata su eventi consente di gestire questi eventi e di eseguire codice specifico in risposta a tali eventi.
20. Che cosa sono lo stage, la scene, il root node e i nodes in JavaFX? In JavaFX, uno stage è una finestra principale dell'applicazione che contiene una o più scene. Una scena rappresenta il contenuto principale dell'interfaccia utente e contiene un albero di nodi. Il root node è il nodo principale dell'albero di nodi della scena, a partire dal quale sono collocati gli altri nodi.

    I nodes in JavaFX sono elementi dell'interfaccia utente, come pulsanti, etichette, caselle di testo, immagini, ecc. I nodes possono essere posizionati all'interno della scena, manipolati per cambiare la loro posizione, dimensione, aspetto e comportamento, e possono essere resi interattivi mediante la programmazione basata su eventi.

21. Che cosa sono le lambda expressions in Java e come si usano in JavaFX? Le lambda expressions in Java sono una funzionalità introdotta in Java 8 che consente di creare oggetti funzionali senza dover definire una classe separata per implementare l'interfaccia funzionale. Le lambda expressions in Java possono essere utilizzate per scrivere codice più conciso e leggibile, riducendo la quantità di boilerplate code necessario per implementare l'interfaccia funzionale.

    In JavaFX, le lambda expressions possono essere utilizzate per gestire gli eventi generati dall'interfaccia utente. Ad esempio, è possibile utilizzare una lambda expression per creare un'azione che viene eseguita quando un pulsante viene cliccato. La sintassi per una lambda expression in Java è la seguente:

    (parametri) -> espressione

    dove i parametri sono gli input della lambda expression e l'espressione è il valore restituito dalla lambda expression. Le lambda expressions in JavaFX vengono utilizzate per associare un gestore di eventi ad un nodo dell'interfaccia utente, come ad esempio un pulsante, attraverso l'uso del metodo setOnAction().