Lösningsförslag: Övningar i binära träd, grafer, etc.

1) En rekursiv datastruktur är en datatyp som innehåller en (eller flera) instans(er) av sig själv.

2) Sökningen i ett balanserat träd är O(logn), eftersom det är ett exempel på divide-and-conquer: vi kan utesluta halva trädet varje gång vi gör ett val att gå till höger eller vänster längs en förgrening.

3) Risken är att ett obalanserat träd kollapsar till en länkad lista. Ett binärt träd bygger på att värden sorteras in under noder med högre värden på ena sidan och lägre värden på den andra, men om traverseringen bara sker längs samma förgrening kommer vi få tidskomplexiteten O(n). Vi får då en datastruktur där alla noder i värsta fall måste traverseras för att hitta värdet.

4) TreeMap och TreeSet använder sig av rödsvarta träd (balanserade träd).

5) En graf är en abstrakt datatyp som gestaltar både datapunkter och förhållandena mellan dem. Datan representeras av noder och förhållanden av kanter mellan dem. Man använder dem ofta i sammanhang där relationen mellan datan är lika viktig att modellera som datan själv.

6) Exempelvis sociala nätverk, GPS-applikationer, datornätverk… ställen som lagrar information om relationer mellan entiteter.

7) Garbage collection är en bakgrundsprocess i Java som rensar upp i heapminnet. När objektreferenser försvinner från callstacken kommer garbage collectorn vid något tillfälle att aktiveras och ta bort de ”döda” objekten som inte längre kan nås.

8) Fördelarna är bland annat att den är är säker: vi kan aldrig hamna out of bounds. Den är också effektiv och kan iterera över alla typer av samlingar, vilket en vanlig for-loop inte kan (den kräver indexering).

9) Nackdelarna är att man inte kan förändra samlingen man itererar över eftersom loopen skapar lokala variabler för värdena.

10) Inorder, Preorder och Postorder. Inorder besöker noderna i en sekvens från lägst till högst värde. Preorder besöker roten först. Postorder besöker undernoderna först och sedan roten.

11) Dijkstras algoritm används för att plotta den kortaste vägen i en graf. Den fungerar enligt idén att den kortaste vägen lokalt = den kortaste vägen globalt. Den fungerar dock enbart så länge kantvikten i grafen inte är negativ (med en negativ kant menas att man tjänar någonting i stället för att betala en kostnad för att följa en kant).

12) Undantag (exceptions) är felobjekt som kastas när någonting går fel i Java. De avbryter då programmet. Man kan hantera dem genom att fånga dem med ett catch-block och logga felinformationen.

13) Stack unwinding är vad som händer med stacken när ett undantag kastas: det rippar genom callstacken och poppar stackframe efter stackframe tills det hittar ett catch-block (om sådant finns) som det fastnar i. Om det inte finns någon felhantering kommer det att spola tillbaka hela stacken och programmet kommer att avslutas med en felkod.

14) Märkta undantag (checked exceptions) är sådana som Java tvingar en att hantera på något sätt för att koden ska gå att kompilera. Ett exempel är filhantering med en BufferedWriter, som antingen kräver att man hanterar ett potentiellt undantag direkt i filinläsningsmetoden eller också markerar att den kan kasta ett undantag och låter anroparen (callsiten) bestämma själv hur denne vill hantera det (om alls).

15) Tvåstegsinitialisering är när man skapar ett objekt där alla variabler inte är initialiserade. Ett exempel är ett personobjekt som har en setter för namnet så att det kan matas in efteråt. Om någon försöker skriva ut namnet för personen utan att först ha tilldelat den ett namn kommer programmet att kasta ett NullPointerException.

16) En girig algoritm gör alltid det lokalt bästa valet (det största numret, den kortaste vägen till närliggande granne i en graf, osv). Den backtrackar aldrig när den tagit ett beslut utan håller fast vid det (envis algoritm!).

Det här innebär att den kan stoppa utan att finna en lösning ifall det inte finns en väg vidare (den fungerar därför inte för t.ex. en djupet först-sökning).