1. **Koja su dva osnovna problema u razvoju software-a?**
   Složenost softwarea (i s tehničke strane i sa strane domene) i mijenjajući zahtjevi.

2. **Usporedba Iterativnog i Waterfall modela procesa razvoja.**
   Waterfall model procesa razvoja softwarea se bazira na jasno odvojenim fazama razvoja – izrada zahtjeva i specifikacije, dizajn sustava, implmentacija i testiranje. Svaka faza se mora u potpunosti dovršiti kako bi sljedeća faza mogla započeti.
   Iterativno model faze razvoja vrti u ciklusima (iteracijama) prilikom čega se u svakoj iteraciji rješava dio problema, a ne cijeli problem kao kod Waterfalla.
   Iterativni model omogućuje brže i lakše reagiranje na promjene zahtjeva/potrebe korisnika – važnost ranog feedbacka. Izbjegavamo Rush to Code i Analysis Paralysis. Od velikog je značaja učinkoviti Refactoring.

3. **Navedite pet karakteristika lošeg dizajna.**
   a. Krutost – sistem se teško mijenja jer svaka promjena povlači mnoge druge promjene u ostalim dijelovima sistema
   b. Krhkost – promjene u sistemu uzrokuju pogreške u dijelovima sistema koji nemaju nikakve konceptualne veze s promijenjenim dijelom
   c. Nepokretnost – sistem se teško razdvaja u komponente koje bi se mogle iskoristiti u drugim sistemima
   d. Nepotrebna složenost – dizajn sadrži infrastrukturu koja ne pruža nikakvu dodatnu korist
   e. Nepotrebno ponavljanje – dizajn sadrži ponavljajuće strukture koje bi se mogle unificirati u jednoj apstrakciji

4. **Što je Domain Driven Design?**
   Kod većine softverskih projekata, fokus mora biti na domeni i njezinoj pripadajućoj logici - gradimo sustav u terminima koncepata iz domene.
   Složeni dizajni domene bi trebali biti zasnovani na modelu. Uvijek moramo napraviti model – jer ne želimo (i ne trebamo) u softveru opisati cijelu domenu. U model stavljamo stvari bitne za naš sustav – apstrakcija

5. **Koja su 3 Keypointa DDD-a?**
   a. Model i implementacija oblikuju jedno drugo
   b. Model predstavlja temelj jezika koji se koristi
   c. Model predstavlja distilirano znanje o domeni

6. **Što je Kohezija i koje su vrste kohezije?**
   Kohezija je mjera međupovezanosti (inter-relatedness) svojstava (atributa i operacija) u javnom sučelju razreda.
   Vrste kohezije:
   - Mixed-instance – razred sa mixed-instance kohezijom ima neka svojstva koja su nedefinirana za neke objekte.
   - Mixed-domain – razred sa mixed-domain kohezijom sadrži element koji direktno opterećuje razred sa ekstrinsičnim* razredom neke druge domene
   - Mixed-role - Razred C sa mixed-role kohezijom sadrži element koji direktno opterećuje razred sa ekstrinsičnim razredom koji leži u istoj domeni kao i C

   * (Razred B je ekstrinsičan razredu A, ako se A može potpuno definirati bez znanja o B)
   * (B je intrinsičan prema A ukoliko B modelira neku karakteristiku inherentnu razredu A)

   Kohezija je povezana s Couplingom (vezivanjem), reusabilityjem, pouzdanošću, sposobnošću održavanja.

7. **Što je Enkapsulacija?**
   Enkapsulacija u biti znači skrivanje nekih detalja iza dobro definirane granice.
   Preciznije, OO enkapsulacija je pakiranje operacija i atributa koji predstavljaju stanje u tip objekta (odnosno klasu) tako da je stanje dostupno ili promjenjivo samo preko sučelja definiranog enkapsulacijom.

8. **Što je Apstrakcija?**
   Pojednostavljeno: apstrakcija = bît objekta
   Apstrakcija označava esencijalne karakteristike objekta koje ga razlikuju od svih drugih vrsta objekata i koje definiraju oštre konceptualne granice objekta".

9. **Što je Polimofrizam?**
   Dinamički polimorfizam je mogućnost mijenjanja ponašanja tokom izvođenja. Konretno, radi se s pokazivačima baznog tipa koji pokazuju na izvedene tipove. Drugim riječima, omogućuje se izvođenje akcija preko istog sučelja s vjerojatno različitim implementacijama (za različite tipove). Odabir operacije koja će se izvesti zna se tek tokom izvođenjem, temeljem konkretnog tipa objekta uz pomoć virtualnih funkcija.

   Statički polimorfizam se razrješava u trenutku prevođenja programa. Primjer su parametrizirani programski kodovi i generici.

10. Što je Apstraktna Klasa, a što Sučelje?

Apstraktni razred sadrži članske varijable i funkcije kao i obični razred, s time da su neke (u nekim jezicima barem jedna) metode neimplementirane/nedovršene i/ili označene apstraktnima (C#: abstract T x(), C++: T x()=0). Ne mogu se stvarati objekti apstratnog razreda, već se mora izvesti poseban razred iz apstrantnoga te se implementirati nedovršene metode.

Sučelje razreda predstavlja sve ono što razred pruža prema van – funkcionalnost koju vide drugi objekti. To je apstraktni razred (malo nespretno korištena riječ „razred“) koji definira skup operacija koje moraju implementirati izvedeni razredi.

Napomena: neki jezici dopuštaju nasljeđivanje samo jednog razreda (apstraktnog ili ne), ali zato više sučelja. U C++ (najboljem jeziku ikada) prolazi sve, kako višestruko nasljeđivanje razreda, tako i implicitna konverzija double->int.

11. Overloading i Overriding – definicija i primjeri.

Overloading – mogućnost definiranja više funkcije istog imena, ali s različitim parametrima (povratni tip se ne uzima u obzir) – prevodioc odlućuje koja se funkcija poziva (statički polimorfizam). Često korišteno kod konstruktora razreda.

Primjer: void Ispis(double num) { … }
```
void Ispis(string str) { … }
…
Ispis(1.3) ispisuje na ekran broj 1.3
Ispis(„AAA“) ispisuje na ekran AAA
```

Overriding – reimplementacija neke (virtualne) funkcije u izvedenom razredu.

Primjer:
```
public class Base
{
   public virtual Ispis(string str) { … } // može biti i abstract, bez implementacije
   // Recimo da ispisuje str na ekran
}
public class Derived : Base
{
   public override Ispis(string str) { … }
   // Recimo da ispisuje str unatrag na ekran
}
…
static void Main()
{
   Base normalni = new Base();
   Base inverzni = new Derived();

   // Imamo tu i malo dinamičkog polimorfizma ☺
   normalni.Ispis(„abc“); // Na ekranu abc
   inverzni.Ispis(„abs“); // Na ekranu cba
}
```

12. **Koje domene razreda postoje i kakve su reuse mogućnosti svake domene?**
    a. Fondacijska – razredi sa najširom mogućom primjenom (osnovni i često korišteni tipovi podataka) – veliki reuse
    b. Arhitekturna – Database, Machine to Machine, Human interface – veliki reuse za razne industrije.
    c. Poslovna – Atribute, Role i Relationship razredi - korisno za aplikacije unutar pojedine industrije; upitan reuse.
    d. Aplikacijska – vrlo specifično za pojedinu aplikaciju – slab reuse.

13. **Demeterovo pravilo.**
    • "Don't talk to strangers"
    • "Only talk to your immediate friends"
    • Govori o tome kome sve objekt smije slati poruke
    Tehnički, objekt bi trebao pozivati servise samo sljedećih objekata:
    – this (odnosno samog sebe)
    – Objekti koji su mu predani kao parametri
    – Objekti koje je sam kreirao
    – Bilo koji objekt na koji ima referencu, a koji je njegov podobjekt

14. **Single Responsibility Principle.**
    Isto što i kohezija (na malo drugačiji način). Razred treba imati samo jedan razlog za promjenu. Drugim riječima, razred treba imati (modelirati) samo jednu odgovornost (responsibility).

15. **Open-Closed Principle.**
    Softverski entiteti (razredi, moduli, funkcije, ...) trebaju biti otvoreni za proširenje (kod promjene zahtjeva, imamo mogućnost proširenja modula sa novim ponašanjem koje zadovoljava te zahtjeve), ali zatvoreni za modifikaciju (proširenje ponašanja modula ne zahtijeva promjene u izvornom ili izvršnom kodu modula). Ovaj princip savjetuje da se sistem promijeni tako da daljnje promjene tog tipa ne uzrokuju nove modifikacije. Zvuči kao ideal (i jest).

16. **Liskov Substitution Principle.**
    Podtipovi moraju biti supstitabilni (umetljivi) umjesto svojih baznih tipova. Ponašanje programa mora ostati nepromijenjeno* kada umjesto objekta baznog razreda podmetnemo objekt izvedenog razreda.

    *U nekim design patternima kao što je Strategy i želimo promijeniti ponašanje programa ako umetnemo drugi tip pod pokazivač na bazni razred. Ono što se zapravo misli je da program i dalje ispravno radi, tj. da se prevede i izvrši bez pogreške te pokazuje željeno ponašanje. To ne znači da Strategy Pattern krši LSP, već samo da se malo drugačije shvatila definicija.

17. **Razlika između Entiteta i Vrijednosnog Objekta?**
    Vrijednosni objekt predstavlja neku konkretnu vrijednost nekog podatka pri čemu nam nije bitan identitet vrijednost, već sama vrijednost. Entitet ima svoj identitet i atribute (koji mogu biti vrijednosni objekti ili drugi entiteti)(može imati i metode). Mijenjanje atributa entiteta ne mijenja njegov identitet.

18. Što je Use Case?

Use case opisuje ponašanje sistema u različitim uvjetima kod reakcije sistema na zahtjev nekog stakeholder-a.
Alternativno: Use case predstavlja skup povezanih scenarija (uspješnih i neuspješnih) koji opisuju učesnikovo korištenje sistema radi postizanja nekog cilja.

Omogućavaju fokusiranje na najvažnije pitanje:
Kako korištenje sistema može osigurati vidljivu vrijednost za korisnika, odnosno ispunjenje njegovih ciljeva?

19. Što su Black-box i White-box Use Caseovi? Navesti razlike i dati primjer.

Black-box use case – ne opisuju unutarnje detalje sistema, njegovih komponenti ili dizajna.
White-box use case – suprotno.
Preporuča se korištenje Black-box use casea.

Black-box	White-box
Sistem zapisuje podatke o prodaji	Sistem zapisuje podatke o prodaji u bazu podataka ... Ili (još gore) Sistem generira SQL INSERT naredbu za ubacivanje ...

20. Struktura i elementi Use Caseova?

Primarno se definiraju u tekstualnom obliku, ali moguć je i zapis u obliku UML dijagrama.
Mogu biti sažeti, „obični“ ili potpuni.

Sastoje se od:
- Stakeholder i interest list
- Preduvjeti i postconditions
- Glavni uspješni scenario (osnovni tijek)
- Ekstenzije (alternativni tijek)
- Specijalni zahtjevi
- Popis varijacija u tehnologiji i podacima

21. Što je Design by Contract?

Način oblikovanja sustava u kojemu se postavljaju preconditions i postconditions.
Pošiljatelj zahtjeva za obradu treba osigurati preconditionse nad poslanim parametrima.
Ukoliko su preconditionsi zadovoljeni, objekt koji obrađuje zahtjev garantira da će nakon izvršenja zahtjeva važiti uvjeti definirani u postconditions. Ukoliko preconditions nisu zadovoljeni, postconditionsi mogu, ali ne moraju biti zadovoljeni, nema garancije.