1. **Co je to objekt, jaké jsou jeho vlastnosti?**

* Entita, obsahující data a chování
* Identita – obsahuje objekt od jiných objektů
* Stav – popisuje datovou složku objektu
* Chování – popisuje metody spojené s protokolem objektu
  + protokol = množina všech zpráv, které lze objektu zaslat a ten je schopný na ně reagovat

1. **Co je to třída, jaký je její vztah k objektům?**

* Šablona, pro vytváření objektů
* Definuje složku stavu a chování objektu
  + instanční proměnné uchovávají data ⇒ stav
  + metody ⇒ chování
* instance – objekt vytvořený podle dané šablony

1. **Co říká substituční princip Liskové? Jaké jsou jeho důsledky?**

* pokud S je podtypem (podtřídou) T, pak instance typu S mohou v programu nahradit instance typu T aniž by byla změněna funkčnost programu

1. **Co je to dědění (obecně) a k čemu ho lze využít?**

* Schopnost vytvořit třídu z jiné třídy
* Dědí se vše kromě konstruktorů a destruktorů
* Objekt může být založen na existujícím objektu
* Umožňuje členění a znovupoužití kódu
* Druhy:
  + Klasické – objekt definován třídou, třída dědí z jiné třídy
  + Mezi objekty – objekt dědí z jiného objektu
  + Nadtřída – třída, ze které se dědí
  + Podtřída – třída, která dědí

1. **Co je to polymorfysmus, Ad-hoc, parametrický a subtypový polymorfismus v C#?**

* Polys + morphé = mnoho + tvar
* Schopnost více objektů reagovat na stejnou zprávu
  + Ad-hoc
    - metoda je schopna pracovat s argumenty různých typů („overloading/přetěžování“ metod/operátorů)
  + Parametrický
    - metoda je psána obecně bez vazby na konkrétní datový typ
    - šablony v C++, generické typy v C# nebo Javě
  + Subtypový
    - schopnost podtypu reagovat na stejné zprávy jako nadtyp
    - implementace pomocí dědičnosti

1. **Co je rozhraní (Interface)?**

* Množina definicí metod a vlastností
* Rozhraní může dědit z rozhraní, ale ne z třídy
* Třída využívající rozhraní musí implementovat vše, co je jím definováno
  + Metody mohou byt virtuální

1. **Co je abstraktní třída?**

* Něco mezi rozhraním a klasickou třídou
* Je v ní deklarována alespoň jedna abstraktní metoda
* Definuje stav a chování
  + Na rozdíl od rozhraní však může pro libovolnou část definovaných prvků obsahovat i implementaci

1. **Jaké jsou konvence pro pojmenování v .NET?**

* namespace – MujHttpNamespace
* assembly/sestavy – MojeVlastniAssembly
* třídy – MojeTřída
* výjimky – MojeException
* rozhraní – ImojeRozhraní
* private / protected proměnné – \_mojeVlastni

1. **Co je namespace, k čemu slouží a co umožňuje?**

* Jmenný prostor sloužící k tomu, aby logicky sdružoval typy, které spolu souvisejí
* Utváří kontext pro pojmenování
* Klauzule „using“

1. **Co jsou bloky? Jak se v C# vytváří komentáře a jaké mají druhy?**

* Bloky lze vnořovat, tvoří hranice proměnným v nich deklarovaným {….}
* Komentáře umožňují vnitřní dokumentaci zdrojového kódu, kompilátor je ignoruje
  + Jednořádkové - // XML - ///
  + Víceřádkové - /\* ……. \*/ XML - /\*\*…….\*/

1. **Jaké druhy proměnných rozlišujeme v C#?**

* Slouží k uchování hodnot a každá proměnná má explicitní typ
  + Třídní
    - Private, public, internal, protected
  + Lokální
  + Parametry

1. **Jaké podmíněné výrazy lze v C# použít? Popište je.**
   * If – podmíněný výraz
   * Switch – větvení podle hodnoty řídící proměnné
2. **Jaké iterační cykly lze v C# použít? Popište je.**
   * Do, While – podmínka
   * For – „inicializace“, „podmínka“, „iterace“
   * Foreach – iterace v enumerovatelné sekvenci
3. **K čemu slouží u třídy konstruktor? Druhy konstruktorů, přetížení (overloading) konstruktorů?**

* Slouží k řízení inicializace objektu
* Většinou nastavují výchozí stav vlastností (lze využívat k naplnění konkrétních hodnot)
* Overloading konstruktorů – více konstruktorů s různými parametry
* Public, priváte, static

1. **Metoda. Přebíjení (Overriding) vs. Přetěžování (overloading).**

* Metoda
  + Část chování objektu
  + Funkce, nebo procedura patřící k objektu
* Přebíjení/overriding
  + možnost změny implementace v podtypu
  + virtual , override
* Přetěžování/overloading
  + více metod stejného názvu s různými parametry

1. **Co je to statická třída? Jak se liší od instanční třídy a co musí splňovat? Jak se používá?**

* Všechny metody, vlastnosti a proměnné musí být taky statické
* Nelze vytvářet instanci

1. **Interface v C#. K čemu se dá využít? Jak se liší od třídy a abstraktní třídy?**

* Množina definicí metod a vlastností
* Implementace rozhraní – děděním a implementací jeho metod a vlastností
* Rozhraní může dědit z rozhraní, ale ne z třídy
* Třída využívající rozhraní musí implementovat vše co je jím definováno
  + Metody mohou být virtuální

**Vlastnost Interface Abstraktní třída**

Mnohonásobná dědičnost Třída může dědit více interfaců Třída může zdědit pouze jednu abstraktní třídu

Defaultní implementace Ne Ano

Třídní atributy a konstanty Ano Ne

Modifikátor přístupu Všechno public Může specifikovat přístup

1. **Co je abstraktní třída, využití?**

* Stanovuje společnou definici základní třídy, kterou může sdílet více odvozených tříd
* Něco mezi rozhraním a klasickou třídou
* Na rozdíl od rozhraní však může pro libovolnou část definovaných prvků obsahovat i implementaci

1. **Přístupové metody (accessory) v C#.**
   * Get – accessor typu získej, vrací data z pole vlastnosti
   * Set – accessor typu nastav, zapisuje data do pole vlastnosti

* Pro účely definice akcesoru typu set je v rozsahu jeho bloku definována proměnná **value** držící data zapisovaná do pole vlastnosti nebo jinak manipulovaná uvnitř asesoru

1. **Co je delegát a k čemu ho lze využít?**

* Základní kameny událostmi řízeného programování
* Datový typ umožňující referencovat metodu, umožňuje multicasting
* Definice delegáta určuje signaturu přiřazené metody (lze přiřadit pouze metody se shodnou signaturou)
* Definuje která metoda je volána, pokud je odpálena událost
* Událost je oznámení, že byla provedena nějaká akce

1. **Výjimky a mechanismy jejich ošetřování.**

* Umožňují ošetřování chyb, oddělení normálního kódu od ošetřování chyb
* Výjimka (exception) představuje takřka jakoukoliv anomální situaci, která může nastat
* Kód detekující chybu hází výjimku (throw), kód ošetřující nastalý stav výjimku (chytá)
* Všechny výjimky jsou instancí System.Exception nebo jejich potomků
* Výjimky definovány v. NF, lze vytvářet vlastní
  + Try/catch – zkus něco udělat a když nastane chyba, tak jí chytni
  + Try/catch/finally – zkus něco udělat a když nastane chyba, tak jí chytni a **VŽDY** udělej co je ve finally
  + Try/finally – zkus něco udělat, ale **VŽDY** udělej co je ve finally (výjimky se vyhodí až finally proběhne)
* Výjimky se chytají zdola nahoru
* Catch jde řetězit

1. **Generika v jazyce C#, obecná definice, jaké druhy máme, k čemu slouží. Typová inference.**

* Umožňuje pracovat s objekty, o kterých nevíme, jakého jsou typu
* Generické třídy (Generické typy)
  + Maximalizují znuvupoužitelnost kódu, typovou bezpečnost a výkon
  + Umožňují využít jednu definici pro několik datových typů
* Generická interface
* Generické metody
  + Umožňují deklaraci metody s typovým parametrem
  + Funguje zde typová inference (odvození typu)
* Delegáty

1. **Co jsou generické typy a jak je lze využít?**

* Umožňují využít jednu definici pro několik datových typů
* Maximalizují znovupoužitelnost kódu, typovou bezpečnost a výkon

1. **Pole vs. Kolekce objektů. Základní charakteristiky a rozdíly obou.**

* Pole
  + množina objektů stejného typu
  + Délka je dána hodnotou nebo proměnnou při deklaraci
  + Po deklaraci nelze délku měnit
* Kolekce
  + Slouží k seskupení dat
  + Klasické i generické třídy kolekcí
  + Většina má stejné rozhraní
    - Queue – pokud potřebujeme FIFO
    - Stack – pokud potřebujeme LIFO
    - ArrayList, StringCollection, List – přístup indexem
    - Hashtable, SortedList, Dictionary – přístup pomocí klíče

1. **Kolekce objektů. K čemu slouží a jak se používají, základní generická rozhraní pro tvorbu kolekcí**

* Slouží k seskupení dat
* Klasické i generické třídy kolekcí
* Většina má stejné rozhraní
  + Queue – pokud potřebujeme FIFO
  + Stack – pokud potřebujeme LIFO
  + ArrayList, StringCollection, List – přístup indexem
  + Hashtable, SortedList, Dictionary – přístup pomocí klíče

1. **Typ Struct, jak se liší od třídy? K čemu se dá využít?**

* Složitější datová struktura (odlehčený objekt)
* Nepodporuje dědění, předávání hodnotou,
* Při uvážlivém použití mohou zefektivnit program (in-line ukládání),
* Při neuvážlivé využití může způsobit problémy (předávání hodnotou = vytvoření kopie objektu)
* Třídy lze využít k reprezentaci dat
* Referenční typ = předávání odkazem

1. **Co je to LINQ? Jak se používá?**

* Dotazovací jazyk uvnitř kódu
* Pracuje s poli a kolekcemi implementujícími IEnumerable<T>
* Pracuje s XML a SQL databázemi

1. **Dotazování pomocí LINQ. Jak obecně vypadá dotaz? Z čeho se skládá?**

* From – specifikace zdroje dat, nad kterým se provádí dotaz
* Where – definice omezení dotazu (podmínka), pokud se vrátí true, je záznam zahrnut do výsledku
* Select – realizuje projekci (omezuje datové složky ve výsledku)
* Group – vázáno s klíčovým slovem by, seskupuje výsledky podle nějakého klíče
* Into – používá se v kombinaci se select, join nebo group k uložení jeho výsledku, výsledek se dá dále zpracovávat
* Orderby – vzestupné seřazení výsledků podle zadaných kritérií, s klíčovým slovek descending řadí sestupně
* Join – používá se ke spojení prvků z různých zdrojů na základě dané podmínky ekvivalence, využití spolu s klíčovými slovy on a equals
* Let – vytvoření lokální proměnné uvnitř výrazu
* Př.: var výsledek = (from clovek in lidi orderby clovek.Prijmeni select clovek;)

1. **Jak lze v C# pracovat s grafikou?**

* GDI+ - základní mechanismus Windows, nativní podpora přímo v .NET
* WPF – akcelerovaná grafika, XAML
* Direct/Direct3D – bývalo dostupné pro .NET v podobě Managed DirectX, poslední verze 9
* Direct2D – akcelerovaná 2D grafika – Windows Vista + 7, není zatím dostupné v C#
* XNA – nativní podpora v C#, XNA Game Studio

1. **GUI prvky ve WinForms. Implementace vlastních prvků, UserControl vs. Control.**

* .NET smart client aplikace – graficky bohaté, snadné nasazení a update, využití sítě, bezpečnější přístup k místním prostředkům
* Form – viditelný „povrch“ sloužící k zobrazení informací uživateli
* Control – jednotlivý prvek UI umožňující vstup nebo výstup dat, uživatelská akce s ovládacím prvkem generuje události
* WinForm aplikace – Form + ovládací prvky+ reakce na události
* UserControl – umožňuje vytvořit vlastní ovládací prvek kombinací existujících
* Control – vlastní ovládací prvky včetně speciálního vykreslování, využití Systém.Drawing

1. **Co jsou návrhové vzory? K čemu slouží? Jak se dělí?**

* Obecné řešení nějakého problému při návrhu programu
* Popis řešení nebo obecná šablona (nejedná se o knihovnu nebo kus kódu)
  + Tvořivé
    - Řeší problémy související s vytvářením objektů v systému
  + Strukturální
    - Zaměřené na uspořádání tříd nebo komponent systému za účelem zpřehlednění kódu
  + Behaviorální
    - Týkají se chování systému

1. **Návrhový vzor Singleton.**

* Nejjednoduší vzor
* Cílem je zajistit, že existuje pouze jedna instance dané třídy a poskytnout metodu, která je zpřístupňuje

1. **Návrhový vzor AbstractFactory.**

* Zapouzdřuje množinu podobných „továren“ vytvářejících rodin objektů
* Při jeho použití není nutné specifikovat danou třídu

1. **Návrhový vzor Adaptér.**

* Mění rozhraní třídy na rozhraní očekávané klientem

1. **Návrhový vzor Dekorátor.**

* Umožňuje objekt dynamicky doplnit o další vlastnosti
* Alternativa k vytváření podtypů

1. **Návrhový vzor Fasáda.**

* Používá se tam, kde je stávající subsystém tříd příliš složitý
* Poskytuje jednotné rozhraní vyšší úrovně ke všem rozhraním tohoto subsystému
* Klient nemusí komunikovat s jednotlivými rozhraními subsystému, ale pouze s fasádou, která zná strukturu subsystému a volání předává

1. **Návrhový vzor Skladba.**

* Umožňuje vytvářet stromovou strukturu skládající se z jednoduchých a z nich složených objektů
* Složený objekt = kolekce objektů, z nichž každý může být jednoduchý nebo znovu složený

1. **Návrhový vzor Stav.**

* Změna chování objektu při změně jeho vnitřního stavu
* Využívá se třídy zapouzdřující rozdílné chování v závislosti na stavu objektu
* Objekt zná svůj aktuální stav a využívá jeho metody
* Aby vše správně fungovalo, musí mít všechny stavy stejné rozhraní
* Neřeší kdo je zodpovědný za určení aktuálního stavu

1. **Návrhový vzor Strategie.**

* Základní podmínkou je existence různých řešení stejného problému
* Objekty mají stejné rozhraní
* Výchozí objekt obsahuje odkaz na některou z konkrétních strategií

1. **Co je refactoring? Proč a kdy ho děláme, k čemu slouží?**

* Techniky úpravy existujícího kódu za účelem změny jeho struktury bez změny funkčnosti
* Motivace = pachy v kódu (duplikovaný kód, dlouhá metoda, velká třída, mnoho parametrů, líná třída – třída, která skoro nic nedělá)
* Refaktorováním můžeme získat – čitelnější kód, možnost rozšíření kódu
* Kód by měl být testován před započetím refaktorování a poté po každém refaktorovacím kroku
* Techniky pro zvětšení abstrakce:
  + Zapouzdření proměnné
  + Náhrada typové kontroly stave nebo strategií
  + Náhrada podmíněného výrazu polymorfismem
* Techniky pro dělení kódu:
  + Vyčlenění metody
  + Vyčlenění třídy
* Techniky ovlivňující podobu kódu
  + Přesun metody nebo proměnné
  + Přejmenování metody nebo proměnné
  + Pull up (přesun metody/proměnné do nadtřídy)
  + Push down (přesun metody/proměnné do podtřídy)

1. **Unit testing, k čemu slouží? Míry pokrytí kódu testy, cyklomatická složitost, PVC.**

* Automatizované testování kódu po částech (jednotkách)
* Jednotka – procedurální programování (modul, častěji ale jen funkce či metoda), objektové programování – obvykle celá třída
* Jednotkové testy připravují sami programátoři
* Cílem je ověřit, že kód dělá, co by měl
* Při změně v kódu umožňuje snadno ověřit, že neovlivnila jiné části
* Jedna testovaná třída by měla vždy testovat jeden objekt/třídu
* Testované třídy by se měly jmenovat podle testované třídy
* Jedna testovací metoda by měla provádět jeden test (test se může skládat z více kroků)
* Jednotkové testy by neměly nijak ovlivňovat systém
  + Testování s využitím aserce
  + Testování vyjímek
* 100% pokrytí kódu = každá metoda má svůj test
* 100% pokrytí **NENÍ** 100% otestování
  + Cyklomatická složitost – počet nezávislých cest algoritmem
  + PVC – jak mohou hodnoty parametrů ovlivňovat výsledek?