**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

**Semestrálna práca 2**

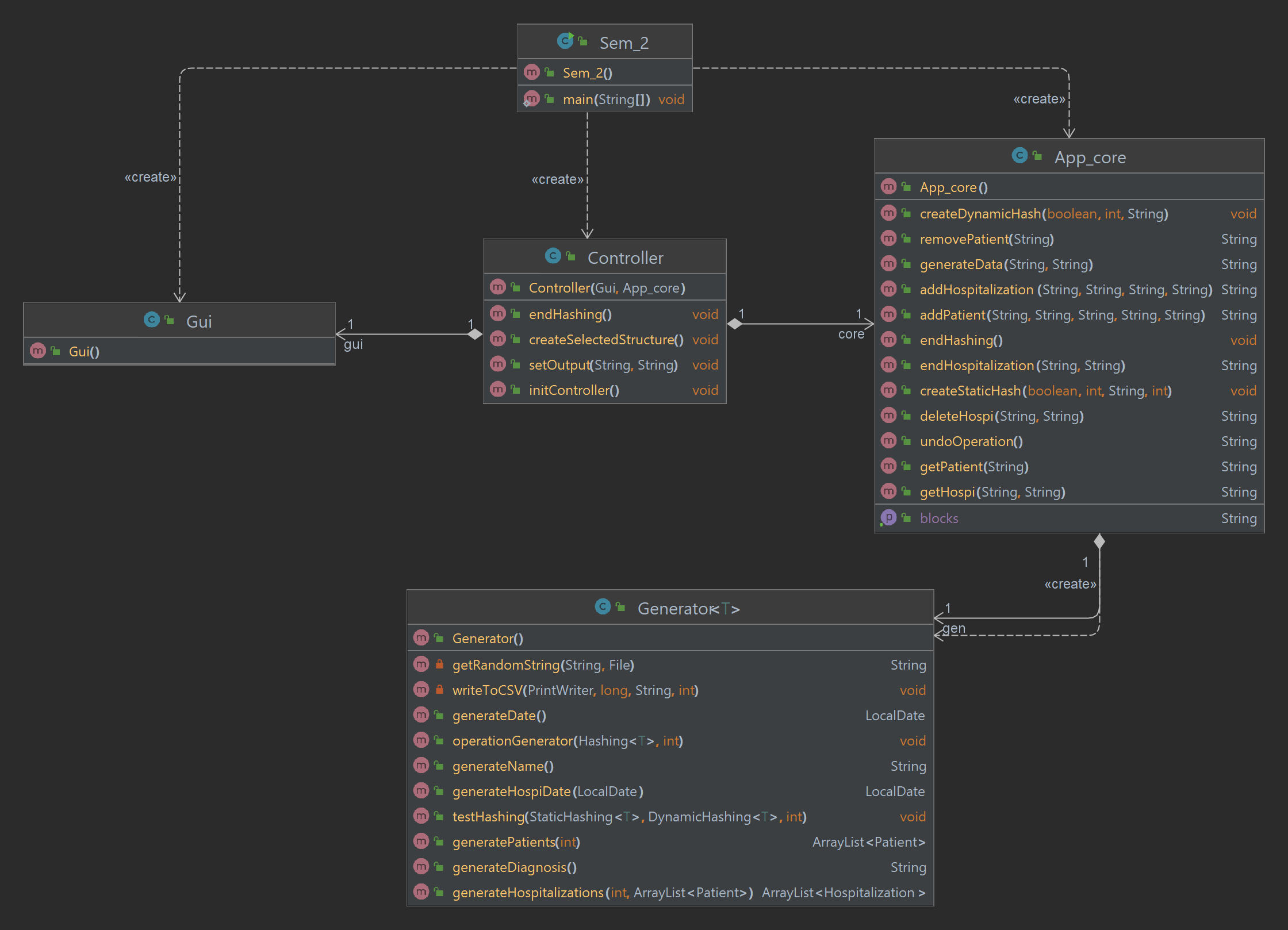
Algoritmy a údajové štruktúry 2

**ŽILINA 2022 FILIP SUDORA**

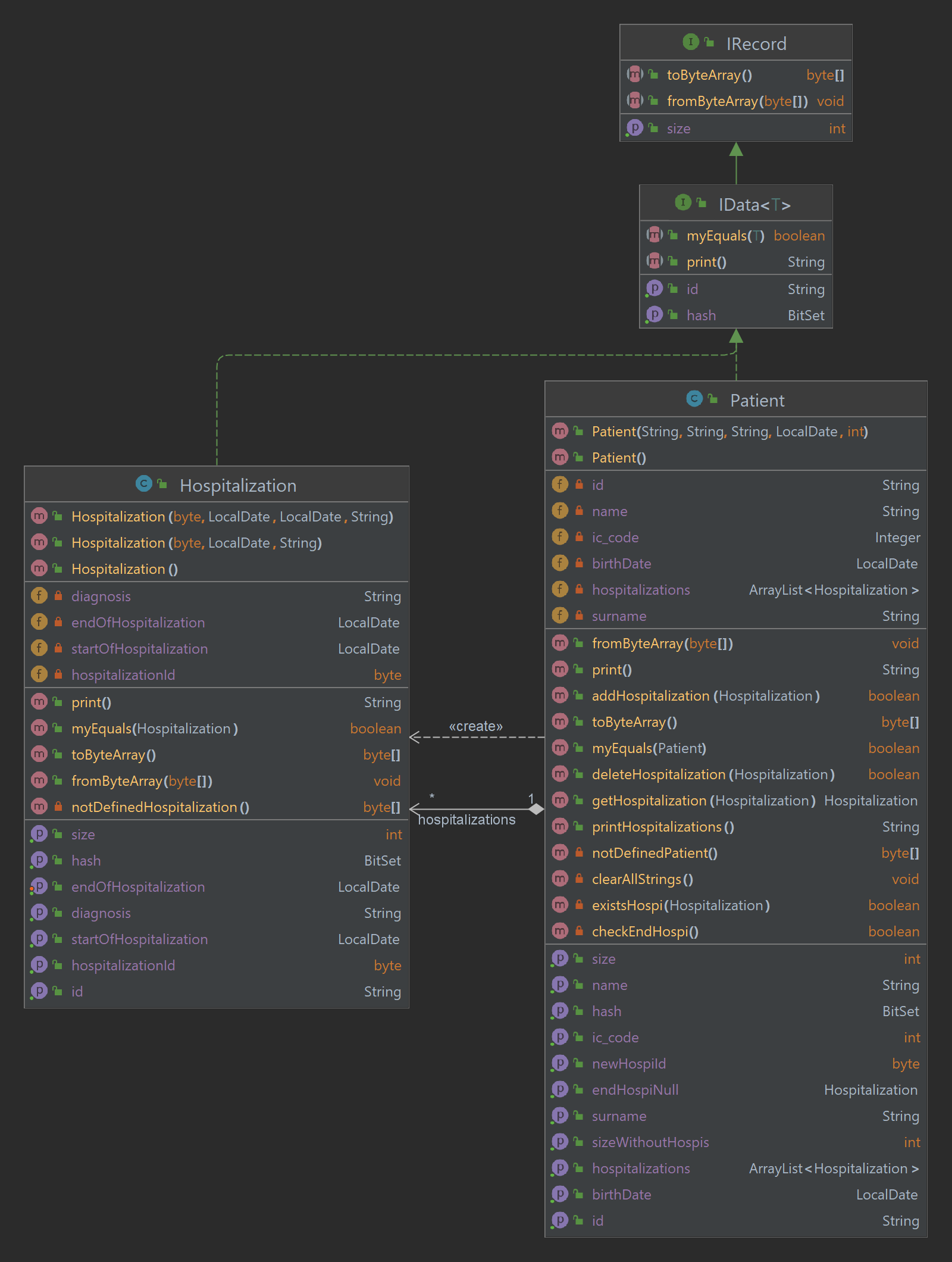
# UML diagram tried

Na nasledujúcom obrázku sú zobrazené triedy zodpovedné za beh aplikácie ako takej.

* Trieda **Gui** - slúži na zobrazenie používateľského rozhrania.
* Trieda **App\_core** - zabezpečuje tvorbu údajových štruktúr a spracovávanie požadovaných operácií informačného systému.
* Trieda **Controller** - zabezpečuje komunikáciu medzi používateľským rozhraním a hlavnou časťou aplikácie.
* Trieda **Generator** - generuje dáta do údajových štruktúr, umožňuje naplniť prázdnu štruktúru z preddefinovaného zoznamu prvkov a obsahuje metódy na testovanie údajovej štruktúry.
* Trieda **Sem\_2** – predstavuje vstupný bod do aplikácie.

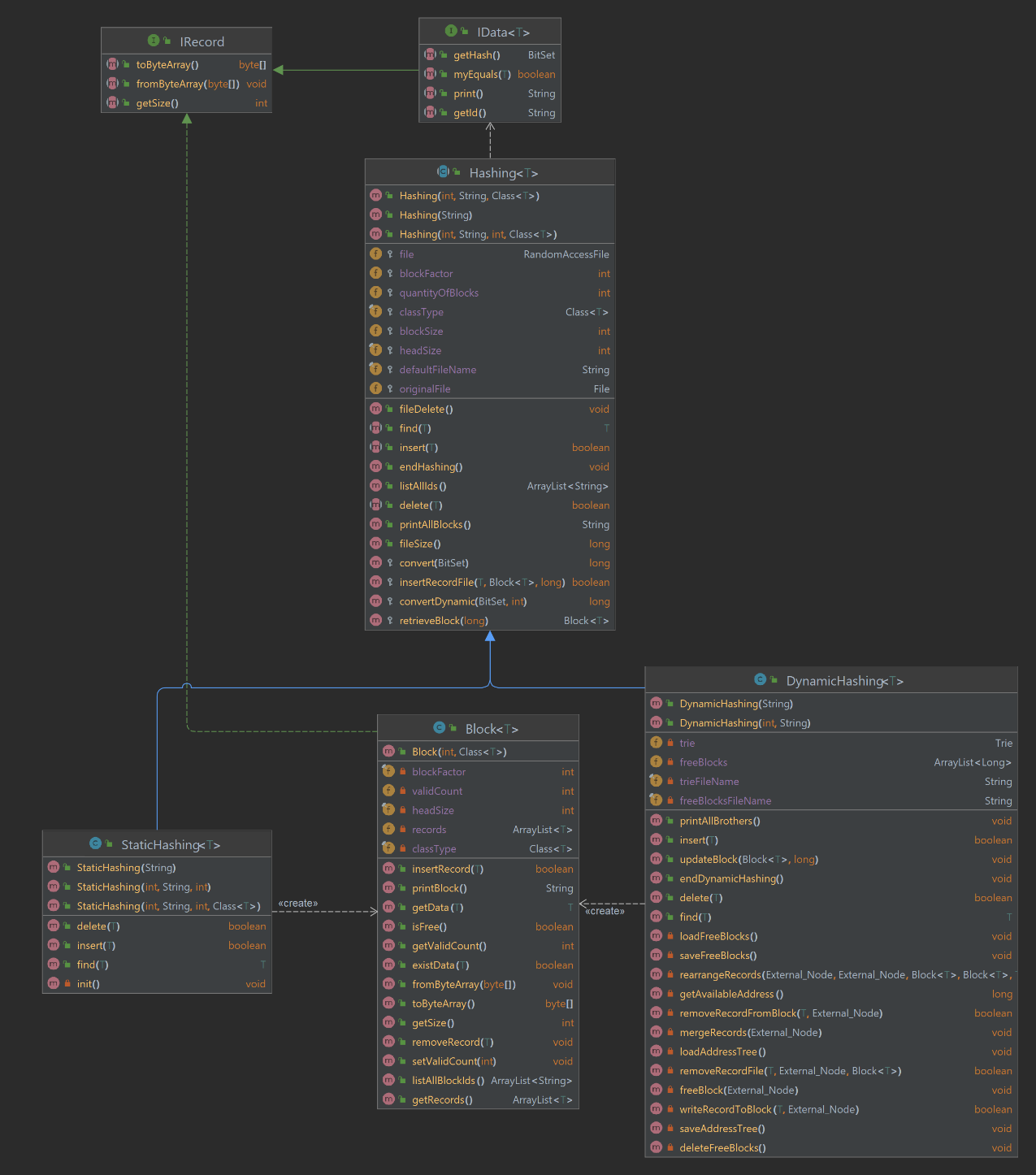


Na nasledujúcom obrázku je zobrazený diagram tried, ktoré uchovávajú dáta o jednotlivých prvkoch systému (**Patient**, **Hospitalization**) a dve rozhrania. Tieto definujú buď metódy na zápis/načítanie zo súboru (**IRecord**) alebo metódy potrebné pri zahešovaní/porovnávaní kľúča (**IData**).



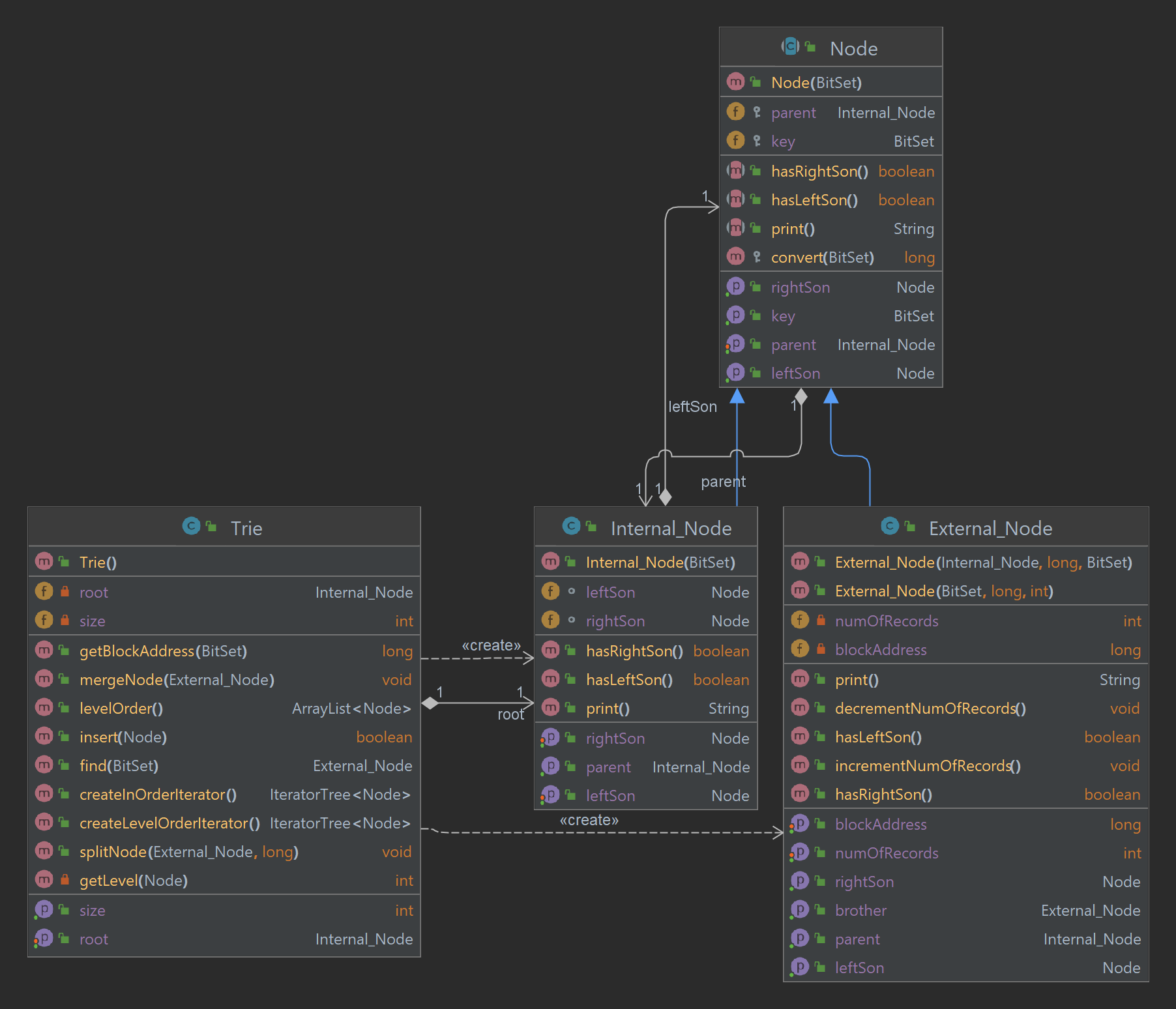
Na nasledujúcom obrázku sú zobrazené triedy, ktoré tvoria implementované typy hešovacích súborov.

* Trieda **Hashing –** predstavuje predka oboch typov hešovacích súborov, ktoré sme implementovali. Vyskytujú sa v nej metódy a atribúty, ktoré sú spoločné pre oba typy súborov.
* Trieda **StaticHashing –** pomocou nej je implementovaný statický hešovací súbor.
* Trieda **DynamicHashing** **–** pomocou nej je implementovaný dynamický hešovací súbor. Je v nej definovaná pomocná stromová štruktúra Trie. Taktiež si uchováva zoznam voľných blokov v súbore.
* Trieda **Block** **–** predstavuje blok, ku ktorému pristupujeme do súboru. Ukladá si potrebné atribúty pre prístup a korektné načítanie bloku zo súboru.



Na nižšie uvedenom obrázku sú zobrazené triedy, ktoré zabezpečujú ukladanie adries do súboru pre dynamický hešovací súbor.

* Trieda **Trie –** pomocou tejto triedy sa získava adresa do súboru pre dynamický hešovací súbor. Taktiež disponuje metódami **mergeNode** a **splitNode** slúžiacimi na zjednotenie/rozdelenie zadaného vrcholu.
* Trieda **Node –** všeobecný predok oboch typov vrcholov v strome **Trie**. Ukladá si odkaz na rodiča.
* Trieda **Internal\_Node** **–** vnútorný vrchol stromu **Trie**. Ukladá sa v ňom referencia na oboch synov.
* Trieda **External\_Node** **–** list stromu **Trie**, v ktorom sa ukladá predovšetkým adresa do súboru.



# Prístupy do súboru

## Statický hešovací súbor

1. Vyhľadanie záznamov pacienta
   1. Blok sa raz načíta vo všetkých prípadoch. (1x)
2. Vyhľadanie hospitalizácie
   1. Blok sa raz načíta vo všetkých prípadoch. (1x)
3. Vykonanie záznamu o začiatku hospitalizácie
   1. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa úspešne pridá hospitalizácia a blok sa zapíše. (2x)
   2. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa nepridá hospitalizácia. (1x)
   3. Nájde sa blok, v ktorom sa má pacient nachádzať, ale pacient sa v bloku nenašiel. (1x)
4. Vykonanie záznamu o ukončení hospitalizácie
   1. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa úspešne ukončí hospitalizácia a blok sa zapíše. (2x)
   2. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa nezmení hospitalizácia. (1x)
   3. Nájde sa blok, v ktorom sa má pacient nachádzať, ale pacient sa v bloku nenašiel. (1x)
5. Pridanie pacienta
   1. Nájde sa blok, v ktorom sa má nachádzať pacient. Pacient sa úspešne pridá a blok sa zapíše. (2x)
   2. Nájde sa blok, v ktorom sa má nachádzať pacient. Pacient sa nepridá do bloku. (1x)
6. Vymazanie hospitalizácie
   1. Načíta sa blok, ale pacient sa nenašiel v bloku. (1x)
   2. Načíta sa blok, ale hospitalizácia sa nenašla v pacientovi. (1x)
   3. Načíta sa blok, pacient aj hospitalizácia sa našli a zároveň sa hospitalizácia sa odstránila. (2x)
7. Vymazanie pacienta
   1. Získa sa blok, ale pacient sa nenašiel v bloku. (1x)
   2. Získa sa blok a pacient sa odstráni z bloku. (2x)

## Dynamický hešovací súbor

1. Vyhľadanie záznamov pacienta
   1. Blok sa raz načíta vo všetkých prípadoch. (1x)
2. Vyhľadanie hospitalizácie
   1. Blok sa raz načíta vo všetkých prípadoch. (1x)
3. Vykonanie záznamu o začiatku hospitalizácie
   1. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa úspešne pridá hospitalizácia a blok sa zapíše. (2x)
   2. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa nepridá hospitalizácia. (1x)
   3. Nájde sa blok, v ktorom sa má pacient nachádzať, ale pacient sa v bloku nenašiel. (1x)
4. Vykonanie záznamu o ukončení hospitalizácie
   1. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa úspešne ukončí hospitalizácia a blok sa zapíše. (2x)
   2. Nájde sa blok, v ktorom sa nachádza pacient. Pacientovi sa nezmení hospitalizácia. (1x)
   3. Nájde sa blok, v ktorom sa má pacient nachádzať, ale pacient sa v bloku nenašiel. (1x)
5. Pridanie pacienta
   1. Vkladaný pacient zmesí do získaného bloku a ešte v ňom neexistuje. (2x)
   2. Vkladaný pacient zmesí do získaného bloku, ale už v ňom existuje. (1x)
   3. Vkladaný pacient sa nezmestí do bloku, načíta sa a skontroluje sa existencia pacienta v bloku – prvý prístup. Následne sa v závislosti od počtu neúspešných opakovaní pokusov o vloženie pacienta do bloku (n) pristupuje v každom opakovaní: jeden zápis ak sa všetky záznamy zostali v aktuálnom bloku alebo dva zápisy ak sa všetky záznamy presunuli do susedného bloku. Na konci sa dvakrát zapíše. (3+ x)
6. Vymazanie hospitalizácie
   1. Načíta sa blok, ale pacient sa nenašiel v bloku. (1x)
   2. Načíta sa blok, ale hospitalizácia sa nenašla v pacientovi. (1x)
   3. Načíta sa blok, pacient aj hospitalizácia sa našli a zároveň sa hospitalizácia sa odstránila. (2x)
7. Vymazanie pacienta
   1. Ak sa nezlučujú susedné bloky a pacient existuje v bloku, tak sa blok raz načíta a raz zapíše. (2x)
   2. Ak pacient neexistuje v bloku, tak sa blok raz načíta. (1x)
   3. Ak sa bloky môžu po odstránení pacienta zlúčiť, tak sa cyklicky v každom zlúčení dvakrát načíta a dvakrát zapíše. (2+ x)

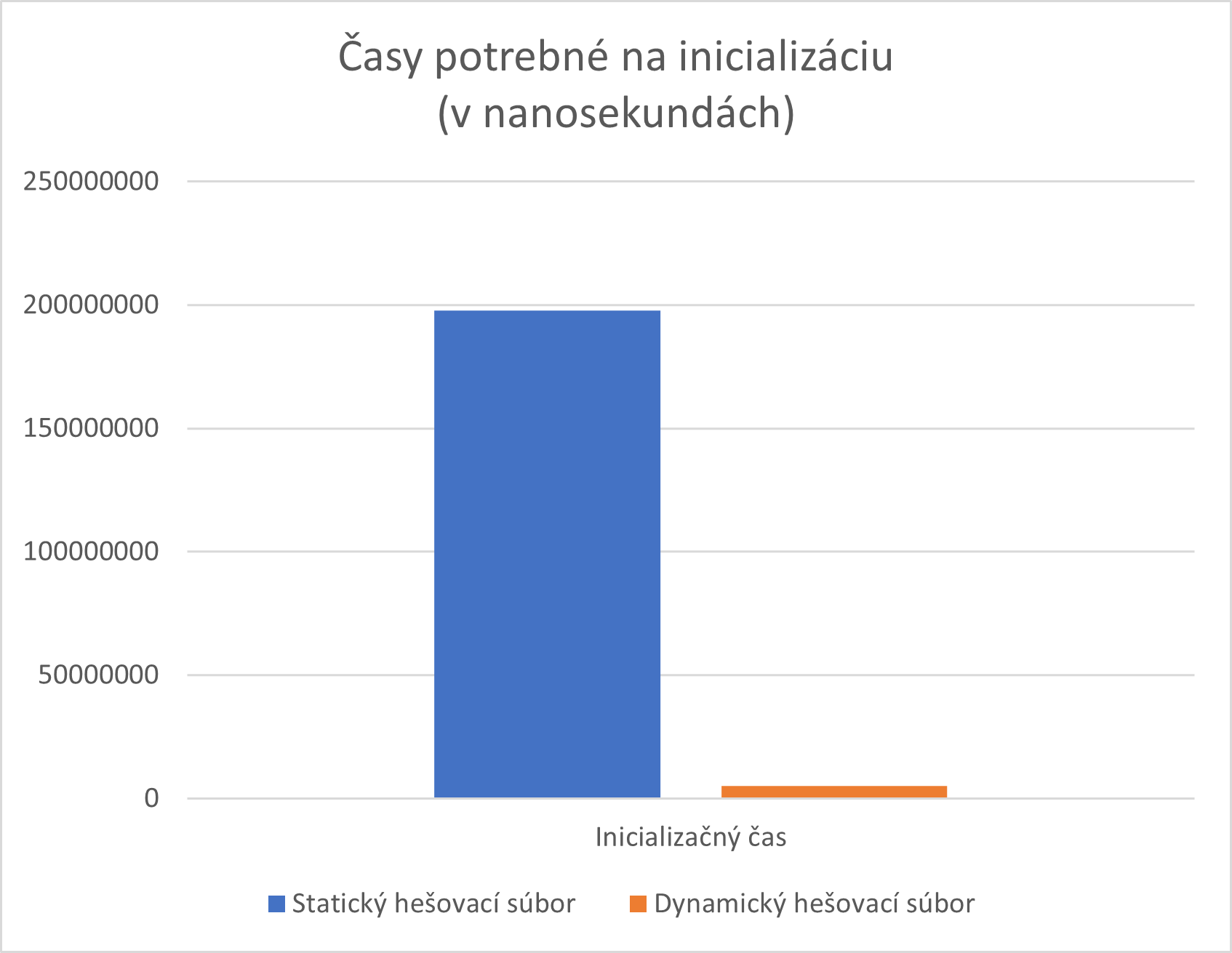
# Porovnanie údajových štruktúr

Používanie aplikácie je rýchlejšie ak používateľ pracuje so statickým hešovacím súborom. A to predovšetkým z dôvodu priameho prístupu k dátam bez nutnosti prechádzania pomocných štruktúr, prípadne upravovania (rozdeľovania/spájania) blokov ako tomu je v dynamickom hešovacom súbore. Avšak, toto so sebou prináša negatívny efekt v podobe fixnej veľkosti súboru, kedy nám veľkosť súboru buď nemusí postačovať alebo naopak príliš mrháme priestorom na disku keď je súbor málo zaplnený. Vyššie spomenutý negatívny efekt je až natoľko výrazný, že rýchlostné nedostatky pri práci s dynamickým hešovacím súborom nie sú zvyčajne príliš dôležité. Jednotlivé rozdiely bližšie rozoberáme v nasledujúcom texte.

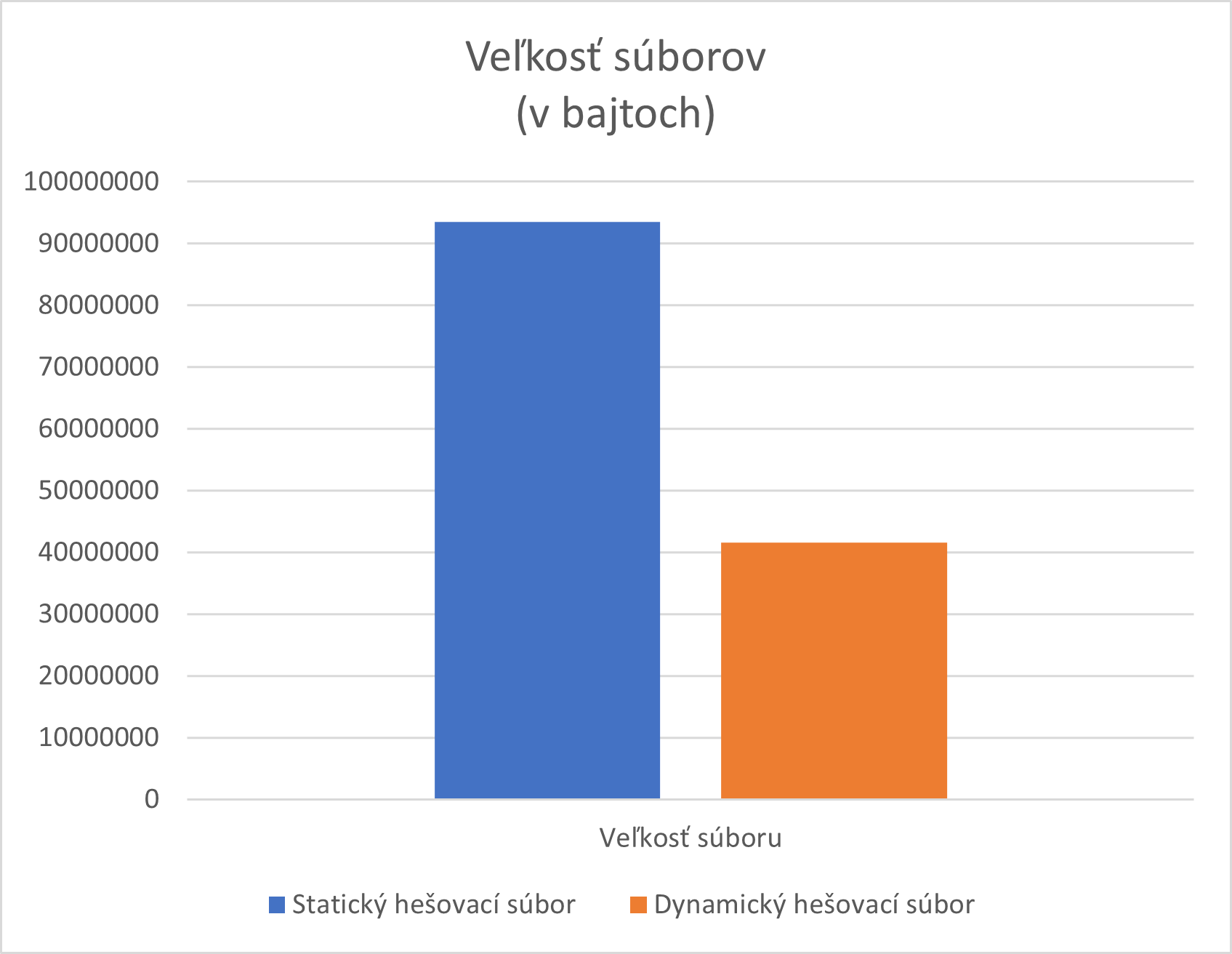
## Výkonové testy

Najväčšie rozdiely pri porovnávaní oboch štruktúr nevznikli pri práci s danými štruktúrami, ale pri porovnaní trvania inicializácie a pri konečnej veľkosti súboru.

Na nasledujúcom obrázku môžeme vidieť inicializačný čas potrebný na pripravenie danej štruktúrou. Čas potrebný na alokovanie pamäte pre 100 000 záznamov s blokovacím faktorom 10 bol pre statický hešovací súbor 197889300 nanosekúnd (0,1979 sekundy), zatiaľ čo pre dynamický hešovací súbor 5039400 nanosekúnd (0,005 sekundy). To predstavuje približne 40 násobné oneskorenie v neprospech statického hešovacieho súboru. Tento rozdiel je spôsobený nutnosťou alokácie pamäťového priestoru na disku pri inicializovaní štruktúry. Pri inicializovaní dynamického hešovania sa alokoval priestor pre 2 bloky, pri statickom 10 000 blokov.



Na nasledujúcom obrázku môžeme vidieť konečnú veľkosť súborov po vykonaní 100 000 operácií (50% vkladanie, 20% hľadanie, 30% mazanie). Konečná veľkosť statického dynamického súboru bola o viac ako dvojnásobok (2,25) väčšia oproti konečnej veľkosti dynamického hešovacieho súboru.



Na ďalšom obrázku porovnávame priemerné trvanie operácií vkladania, hľadania a mazania nad statickým a dynamickým hešovacím súborom.

Najmenšie rozdiely medzi štruktúrami vznikli pri operácií hľadania. Pri vykonávaní tejto operácie nad dynamickým hešovacím súborom sa najskôr nájsť index/adresa do súboru v pomocnej údajovej štruktúre (Trie). V statickom hešovacom súbore sa pristupuje na adresu priamo (bez pomocnej štruktúry). Pri operáciách **Vlož** a **Vymaž** sú už rozdiely znateľnejšie. Taktiež sa musí nájsť adresa do súboru prostredníctvom pomocnej údajovej štruktúry (Trie), avšak pri určitých situáciách je nutné pri vkladaní rozdeľovať blok a pri mazaní spájať dva bloky do jedného (vo výnimočných prípadoch aj cyklicky), čo má za následok dlhšiu priemernú dĺžku trvania týchto dvoch operácií.

