CodEx – Příručka běžného uživatele

Obsah

1. Úvod

Uživatelské rozhraní má podobu webové aplikace. Pro přístup do CodExu doporučujeme používat moderní prohlížeče podporující běžné standardy W3C (např. FireFox, Opera, SeaMonkey, …) s podporou JavaScriptu. Mobilní zařízení a netradiční prohlížeče nejsou podporovány a aplikace v nich nemusí fungovat správně.

2. Přihlášení a registrace

Přístup do aplikace je zabezpečen autorizačním formulářem, kde je nutno zadat správnou kombinaci přihlašovacího jména a hesla. Po prvním přihlášení do aplikace si nejprve na stránce **osobní nastavení** zkontrolujte parametry svého účtu. Pokud máte navíc heslo zadané správcem, změňte si je.

Pozn: Heslo se do databáze ukládá pouze v hašované podobě, takže jej nemůže nikdo vidět (ani správce). Rovněž není možné heslo zjistit v případě jeho ztráty, avšak správce vám může vygenerovat heslo nové.

Studenti MFF UK mohou využít možnosti automatické **registrace**. Jejím prvním krokem je zadání přihlašovacích údajů do Informačního Systému Studium UK. Po ověření v ISu je student vpuštěn na registrační stránku, kde může vyplnit parametry nově vytvářeného uživatelského účtu.

Ostatní uživatelé (zejména pak ti s vyššími právy) musí být do systému zadáni jiným uživatelem, který má dostatečné oprávnění.

3. Osobní nastavení

Stránka osobní nastavení slouží k modifikaci údajů přihlášeného uživatele. Pokud má uživatel delegovaná nějaká práva, zobrazí se zde jejich přehled.

3.1 Přihlašovací jméno a alias

Přihlašovací jméno slouží nejen k přihlašování, ale zároveň jako jednoznačná identifikace uživatele. V přihlašovacím jméně se nerozlišují malá a velká písmena. V nasazení pro MFF předpokládáme, že přihlašovací jméno odpovídá osobnímu číslu studenta. U cvičících a jiných uživatelů, kteří toto číslo nemají, je zpravidla přihlašovací jméno shodné s příjmením daného uživatele bez diakritiky.

Aby si uživatel nemusel pamatoval přihlašovací jméno přidělené systémem, může si navíc sám nastavit přihlašovací alias. Tento alias pak může při přihlašování vyplňovat místo přihlašovacího jména (stále však funguje možnost přihlásit se pod původním přihlašovacím jménem).

Dojde-li ke kolizi uživatelského aliasu a jiného přihlašovacího jména, které se právě systém pokouší přiřadit, je tento alias smazán a uživatel je o jeho odstranění informován.

3.2 Změna hesla

Při změně hesla je třeba zadat nové heslo dvakrát. Tím se předchází chybnému zadání z důvodu překlepu. Vybírejte dostatečně bezpečná hesla, aby nebylo možné je snadno odvodit ze znalosti informací o uživateli. Uživatel nese odpovědnost za možné následky při prolomení slabého hesla a následný průnik do systému.

3.3 Automatická upozornění

Uživatel si může nechat zasílat upozornění o některých událostech přes e-mail. Upozornění jsou zasílána na adresu, kterou má uživatel vyplněnou v CodExu. Tato služba není garantovaná, neboť ani doručení e-mailu není zaručené a pracuje na principu best-effort. Význam jednotlivých typů upozornění je následující:

- *Nové globální novinky viditelné pro všechny* – Upozornění je zasláno vždy, když do systému přibude novinka viditelná pro všechny uživatele. Viz novinky.
- *Nové novinky pro konkrétní skupiny* – Upozornění je zasláno, když je vytvořena novinka určená pro některou ze skupin, v níž je uživatel členem. Viz novinky.

- *Nová zadaná úloha ve skupině* – Upozornění je zasláno, pokud je do některé skupiny, jíž je uživatel členem, zadána nová úloha.
- *Změna některé úlohy* – Upozornění je zasláno, pokud se změní parametry některé úlohy, kterou má uživatel právo vidět (ať už jako zadanou úlohu ve skupině, nebo proto, že má potřebná oprávnění).
- *Vyhodnocení mého řešení* – Upozornění je zasláno, když skončí vyhodnocování řešení, odevzdaného uživatelem. Toto upozornění používejte jen ve zvláštních případech. Za normálních okolností by nemělo vyhodnocení úlohy trvat déle než desítky vteřin až jednotky minut, a proto je výhodnější si na výsledky počkat přímo v aplikaci.
- *Změna stavu členství ve skupině* – Upozornění je zasláno, když je uživatel přiřazen do nebo vyřazen ze skupiny bez vlastního přičinění.

3.4 Delegovaná práva

Pokud má uživatel delegovaná práva k nějaké skupině nebo úloze, uvidí přehled těchto práv pod formulářem osobního nastavení. Uživatel se může těchto práv vzdát, ale pokud tak učiní, nemůže práva znovu získat, pokud mu je jiný uživatel opět nedeleguje.

4. Novinky

Novinky fungují jako krátké zprávy týkající se aplikace nebo aktivit ve skupinách, které vypisují správci a operátoři. Novinky existují dvou typů:

- *Globální novinky* – Novinky viditelné pro všechny uživatele. Tyto novinky zpravidla vypisuje správce a slouží zejména k oznamování důležitých informací ohledně běhu aplikace (odstávky, aktualizace, závažné chyby apod.)
- *Novinky určené konkrétním skupinám* – Novinky, které jsou viditelné pouze pro členy a operátory určité skupiny. Tyto novinky slouží zejména k informování uživatelů o zadaných úložkách a jiných aktivitách vztahujících se přímo ke skupině.

Věnujte prosím novinkám zvýšenou pozornost, neboť se jedná o základní způsob informování uživatelů. V případě potřeby si nastavte zasílání upomínek e-mailem.

5. Skupiny

Skupiny slouží ke sdružování uživatelů, kteří mají za úkol řešit stejné úlohy. Aby mohl uživatel řešit, musí se nejprve stát *členem* nějaké skupiny. V této skupině pak může řešit *zadané úlohy, z*a které dostane body.

Na stránce skupiny se zobrazují všechny skupiny, které má právo přihlášený uživatel vidět. Klepnutím na jméno skupiny je možné do ní vstoupit.

5.1 Členství ve skupině

Skupiny jsou rozděleny na dva typy, *veřejné* a *privátní*. Veřejné skupiny jsou viditelné pro všechny uživatele a každý se do nich může *přihlásit*. Privátní skupiny nejsou viditelné pro běžné uživatele a přiřazovat do nich členy může pouze operátor této skupiny.

Uživatel, který se stal členem nějaké skupiny (veřejné či privátní), se z této skupiny nemůže sám odhlásit, ale může být ze skupiny vyřazen operátorem.

5.2 Body a plnění povinností

Za vyřešené úlohy je uživatel odměněn body. Body se v rámci skupiny sčítají a mohou sloužit jako hodnocení uživatelů (např. k sestavení pořadí v soutěžích nebo zjištění, zda student získal dostatek bodů na zápočet).

Skupina může mít nastaven *bodový limit*. Uživatel musí získat minimálně počet bodů daný bodovým limitem a vyřešit všechny povinné úlohy, aby *splnil požadavky skupiny*. Požadavky skupiny odpovídají např. nároku na zápočet nebo kvalifikace pro postup do dalšího kola apod.

Pokud je skupina *diskrétní*, uživatel vidí pouze své výsledky a neví nic o ostatních uživatelích. To je vhodné například pro hodnocení uživatelů při výuce, kdy je potřeba dát pozor na ochranu osobních informací. Naopak ve skupině, která není diskrétní, vidí všichni uživatelé výsledky všech ostatních, což může být motivující například u veřejných soutěží.

5.3 Zadané úlohy

Na stránce „**Zadané úlohy**“ naleznete seznam úloh, které jsou ve zvolené skupině zadané a které je možné řešit. U každé úlohy jsou zobrazeny následující údaje:

- *Termín* – datum (a čas), do kdy je možné úlohu odevzdávat za body. Úlohu je možné řešit i po termínu, ale uživatel za ni již žádné body nedostane. Pokud je uvedeno pouze datum bez času, je termínem půlnoc, kterou tento den začíná. Např. 14.4. znamená ve skutečnosti 14.4. (0:00), takže řešení odevzdaná už jen jednu vteřinu po půlnoci jsou považována za opožděná). Dále viz níže.
- *Přidělené body* – počet bodů, které uživatel získal (za nejlepší odevzdané řešení) a maximální možný počet bodů. Body se určí jako poměrná část z maximálního počtu bodů dle procentuální správnosti řešení.
- *Povinné body* – minimální počet bodů, které musí uživatel získat, aby splnil danou povinnou úlohu. Pokud je počet roven nule, úloha není povinná.
- *Minimální správnost* – požadovaná správnost (vyjádřená v promille) řešení, aby bylo uznáno a počítaly se z něj body. Tento limit funguje jako ochrana před řešeními, která se snaží např. náhodně tipovat výsledky.
- *Počet odevzdaných řešení* – počet odevzdaných řešení a limit na počet pokusů. Pokud uživatel vyčerpá počet pokusů, nemůže odevzdat další řešení.
- *Přípony* – seznam povolených jazyků, ve kterých je možné úlohu řešit. Každý jazyk je jednoznačně určen příponou, která se používá pro příslušný typ souborů. Přípona také určuje, jaký kompilátor a jaké jeho nastavení se použije pro překlad řešení.

Klepnutím na název zadané úlohy otevřete její zadání. Zadání je třeba prostudovat velmi pečlivě a přesně jej dodržet. Velkou pozornost věnujte zejména formátu vstupu a výstupu a názvům souborů.

Úloha může mít nastaveny až dva termíny se dvěma různými bodovými limity. Řešení odevzdaná před prvním termínem jsou ohodnocena dle prvního bodového limitu. Řešení odevzdaná po prvním a před druhým termínem jsou ohodnocena dle druhého limitu. Odevzdávat je možné i po 2. termínu, avšak taková řešení jsou bodována 0 body.

5.4 Odevzdání řešení a výsledky

Nové řešení je možné odevzdávat na stránce „**Odevzdat řešení**“, kde se nachází formulář umožňující natažení (upload) řešení ze souboru, nebo přímé vložení přes textové pole. Vřele doporučujeme řešení před odevzdáním zkontrolovat a otestovat alespoň na jednoduchých vstupech. CodEx není určen k ladění vašich řešení a na počet odevzdání je nastaven poměrně nízký limit.

Při testování řešení je nejlepší používat stejné překladače, jaké používá CodEx. Zároveň není rozumné používat ve zdrojových souborech znaky národních abeced (diakritiku) a je vhodné držet se čistého ASCII.

Výsledky všech odevzdaných řešení naleznete na stránce „**Odevzdaná řešení**“. Pokud řešení ještě nebylo vyhodnoceno, není vyplněn čas konce vyhodnocování a místo bodů a správnosti se zobrazuje symbol „?“ (otazník).

Po vyhodnocení se tyto údaje doplní. Body a správnost řešení jsou vyjádřeny číselně, ale pokud řešení neprošlo kompilací, zobrazuje se místo čísla symbol „–“ (pomlčka).

Kliknutím na odkaz „**Info**“ zobrazíte podrobnější informace o odevzdaném řešení. Jedná se především o záznam vyhodnocovacího jádra, který vám může případně poskytnout bližší informace o tom, proč vaše řešení selhalo (viz příloha A).

5.5 Bonusové body

CodEx rozeznává 3 typy bonusových bodů:

- bonusové body řešení
- bonusové body úlohy
- další bonusové body (typicky označované „bonusové body“ bez dalšího přívlastku)

Bonusové body řešení jsou přidány konkrétnímu jednomu odevzdanému řešení a přičítají se k bodům, které uživatel za toto řešení dostal. Tyto body se zobrazují např. na stránce „**Odevzdaná řešení**“ v závorce za normálními body.

Bonusové body úlohy jsou přidány jednomu uživateli za jednu konkrétní zadanou úlohu. Tyto body se sčítají s body za nejlepším odevzdané řešení pro danou úlohu a jsou zobrazeny např. na stránce „**Výsledky**“ skupiny jako další číslo v závorce.

Kromě bodů získaných za odevzdaná řešení vám může operátor skupiny přidělit obecné bonusové body. Bonusové body mohou být uděleny za věci, které nesouvisí s CodExem, ale z nějakého důvodu je vhodné udržovat tyto body pohromadě s body za úlohy. Typickým příkladem použití je vedení docházky nebo hodnocení neprogramátorských úloh.

Bonusové body se přičítají k bodům ve skupině a ovlivňují hodnocení (zda má uživatel splněné povinnosti). Bonusové body mohou být i záporné (tzn. vyjadřovat penále).

6. Řešení problémů

V případě běžných problémů (zapomenuté heslo apod.) se nejprve obraťte na některého z operátorů skupin, ve kterých máte členství. Až v případě vážnějších problémů kontaktujte správce.

Pokud narazíte na zjevnou chybu v aplikaci (např. nefunguje nějaká funkce), můžete tuto chybu nahlásit pomocí odkazu v pravém dolním rohu každé stránky. Pokud je to možné, použijte odkaz z té stránky, na které jste chybu objevili.

Přejeme hodně štěstí při řešení úloh.

Příloha A – Záznam o vyhodnocení

Záznam o vyhodnocení (též log) je souhrn výsledků vyhodnocovacího systému. Jsou v něm obsaženy informace o tom, jak dopadly jednotlivé testy odevzdaného řešení. Log může vypadat například následovně:

```
Jailing user 'codex' (UID 1003, GID 1003) into directory '/home/...'
Initializing... OK
Preparing sandbox... running locally (INSECURE), OK
Finding source... ./inbox/source.pas
Compiling... OK
Test 1... <init> <run> <filter> <check> OK:OK (200 points)
Test 2... <init> <run> <filter> <check> OK:OK (200 points)
Test 3... <init> <run> <filter> <check> OK:OK (600 points)
```

První čtyři řádky nejsou uživatelsky zajímavé a týkají se pouze vnitřního vyhodnocovacího systému. Pokud nastane chyba na některém z těchto řádků, je pravděpodobně chyba v zadání úlohy.

Pátý řádek obsahuje stav kompilace. Pokud je vše vpořádku, objeví se zde:

```
Compiling... OK
```

V opačném případě se objeví hlášení:

```
Compiling... FAILED:
```

Následované chybovým výpisem kompilátoru. Všechny testy pak skončí s hlášením "`Compile error`".

Pokud kompilace dopadla dobře následují výsledky jednotlivých testů. Pokud odevzdané řešení prošlo daným testem, objeví se u testu hláška OK a za ní následuje počet promille bodů, které byly za tento test přiděleny.

```
Test 1... <init> <run> <filter> <check> OK:OK (200 points)
```

Pokud test dopadne špatně, zobrazí se zde chybová hláška a žádné body přiděleny nejsou. Např.:

```
Test 1... <init> <run> SG:Caught fatal signal 11 (SIGSEGV)
```

A.1 Seznam chybových hlášek

Obecné hlášky

- **`OK (X points)`** — Test dopadl úspěšně. Bylo vám přiděleno `X` promille bodů.
- **`Wrong answer`** — Odpověď testovaného programu neodpovídá očekávanému správnému výsledku.
- **`No output file`** — Testovaný program nevytvořil očekávaný výstupní soubor. Zkontrolujte názvy vytvářených souborů.
- **`Time limit exceeded`** — Program překročil povolený časový limit. Příčinou může být volba neefektivního algoritmu, nebo chyba v programu, která způsobí nekonečný cyklus.

- **`Time limit exceeded (after exit)`** — Program překročil povolený časový limit, ale bylo to jen o chlup.
- **`Caught fatal signal 8 (SIGFPE)`** — Nastala chyba při matematické operaci. Typickým příkladem může být celočíselné dělení nulou.
- **`Caught fatal signal 9 (SIGKILL)`** — Váš program byl zabit vyhodnocovacím frameworkem, protože provedl nepovolenou operaci. Za nepovolené operace se považuje vše, co nesouvisí s řešením zadané úlohy (např. speciální systémová volání).
- **`Caught fatal signal 9 during startup (SIGKILL)`** — Program byl zastaven vyhodnocovacím systémem ještě, než došlo ke spuštění jakéhokoli uživatelského kódu. Tento problém nastává, pokud statické proměnné spotřebují více paměti, než kolik má program přiděleno.
- **`Caught fatal signal 11 (SIGSEGV)`** — Program porušil ochranu paměti a operační systém ho musel zabít. Tato chyba zpravidla nastává při:
  - dereferencování **`null`**ového/**`nil`**ového ukazatele
  - přístup do paměti, která není alokována
  - přístup na příliš vzdálené položky pole (tj. na položky, které nejsou alokovány)
  - přetečení zásobníku (např. nekonečnou rekurzí)
  - pokud selže alokace paměti je vrácen **`null`**ový/**`nil`**ový ukazatel (a jeho následná dereference způsobí chybu)
- **`Exited with error status X`** — Vaše aplikace vrátila nenulovou návratovou hodnotu X. To mohlo být způsobeno jednak běhovou chybou (kód chyby závisí na použitém kompilátoru), nebo jste úmyslně vrátili jinou hodnotu, než 0. V jazycích C a C++ je návratová hodnota určena číslem vráceným funkcí main. V jazyce Pascal je návratová hodnota implicitně 0, pokud nepoužijete fukci **`halt`** s návratovou hodnotou.

Hlášky specifické pro Pascal (přípona "`pas`")

- **`Exited with error status X`** — V programu nastala běhová chyba. Některé kódy obsahují i chybovou hlášku. Podrobnosti naleznete v uživatelské dokumentaci Free Pascalu (ftp://ftp.freepascal.org/pub/fpc/docs-pdf/user.pdf, příloha D).

Hlášky specifické pro jazyk C++ (přípona "`cpp`")

- **`Committed suicide by signal 6 (SIGABRT)`** — Tuto chybu způsobí neodchycená výjimka.

Hlášky specifické pro jazyk C# (přípona "`cs`")

- **Problémy s kompilací** — Jazyk C# je v CodExu kompilován a spouštěn v prostředí Mono. Mezi Monem a .NETem od Microsoftu jsou jisté rozdíly. Mono je téměř plně kompatibilní s .NETem 2.0, avšak finesy z novějších verzí neumí. Narazíte-li proto na třídu, kterou vám nechce Mono zkompilovat, budete se muset ve vašem řešení obejít bez ní.
- **`Runtime error 101: Unhandled exception`** — Tento kód je vrácen automaticky, pokud se v programu vyskytla neošetřená výjimka. CodEx obsahuje vestavěný framework pro jazyk C#, který odchytává většinu výjimek a nastavuje i jiné návratové hodnoty (viz dále), avšak framework umí odchytit pouze výjimku hozenou z metody Main (nebo metody tranzitivně zavolané z Main). Výjimky z jiných míst (statický konstruktor, finalizér apod.) se neodchytí, a tím pádem vyústí v návratový kód 101.

- **`Runtime error 102: Dereferencing null value`** — Neošetřená výjimka `NullReferenceException`. Tato výjimka je hozena, pokud se pokusíte dereferencovat proměnnou s hodnotou `null`.
- **`Runtime error 103: Out of memory`** — Neošetřená výjimka `OutOfMemoryException`. Tato výjimnka může je při alokaci nových objektů, pokud už není dostatek místa na haldě. Pozor: Alokace statických objektů inicializovaných ještě před vstupem do metody main může vést také k hození výše zmíněné výjimky, avšak takto hozená výjimka není ošetřena vnitřním frameworkem a proto vede k návratové hodnotě `Runtime error 101: Unhandled exception`.
- **`Runtime error 104: Index out of range`** — Neošetřená výjimka `IndexOutOfRangeException`. Tato výjimka je hozena při přístupu na prvky pole, které jsou mimo alokovaný rozsah.
- **`Runtime error 105: Numeric overflow`** — Neošetřená výjimka `OverflowException`. Tato výjimka je hozena v případě přetečení aritmetických operací uvnitř `checked` výrazu.
- **`Runtime error 106: IO error`** - Neošetřená výjimka `IOException`. Tato výjimka je hozena při chybné IO operaci (např. pokus o čtení za koncem souboru apd.).
- **`Runtime error 107: File not found`** — Neošetřená výjimka `FileNotFoundException`. Tato výjimka je hozena, pokud se pokoušíte otevřít neexistující soubor a nechcete jej vytvořit.
- **`Runtime error 108: Invalid operation`** — Neošetřená výjimka `InvalidOperationException`. Tuto výjimku obvykle hází knihovní objekty, pokud na nich zavoláte metodu, kterou nelze provést vzhledem k aktuálnímu stavu objektu. Například pokud máte prázdný zásobník `Stack` a zavoláte na něj `Pop()`, zásobník vám nevrátí žádný prvek a hodí po vás výjimku `InvalidOperationException`.
- **`Runtime error 109: Division by zero`** — Neošetřená výjimka `DivideByZeroException`. Tato výjimka je hozena při celočíselném dělení nulou.
- **`Caught fatal signal 11 (SIGSEGV)`** — Segmentation fault nastává v Monu pouze pokud přeteče zásobník. Pravděpodobnou příčínou je nekonečná rekurze.
- **`Runtime error 1: User error`** — Uživatelskou chybou skončí aplikace, pokud z metody main vrátíte nenulovou návratovou hodnotu.
- **`Runtime error 42: Using multiple threads`** — Tato chyba nastane, pokud se ve vaší aplikaci pokoušíte pracovat s více vlákny. Vlákna jsou v CodExu zakázána.
- **`Committed suicide by signal 6 (SIGABRT)`** — Signál 6 je poslán aplikaci, pokud jsou paměťové limity nastaveny tak nízko, že se do nich nevejde ani běhové prostředí s programem. Tato chyba typicky znamená, že autor úlohy nastavil příliš nízké paměťové limity pro jazyk C#.

Příloha B – Třída `Reader` pro jazyk C#

Na rozdíl od starších jazyků (Pascal, C ...) nemá C# pěkné knihovní funkce pro práci s textovým vstupem. Aby měli uživatelé i v C# patřičné pohodlí, které nabízel např. **`read`** v Pascalu nebo **`scanf`** v Cčku, byla do CodExu implementována třída **`Reader`**, která usnadňuje načítaní čísel a textových tokenů z konzole nebo textového souboru.

Zdrojový kód třídy si můžete stáhnout z úvodní stránky. CodEx také automaticky linkuje všechna odevzdaná řešení společně s knihovnou obsahující třídu **Reader**, takže je možné ji používat, aniž by bylo nezbytné vkládat její kód do odevzdávaného řešení.

Třída je navržena pouze pro práci se soubory v kódování ASCII. S jiným kódováním (např. UTF-8) si neporadí.

B.1 Způsob použití

Třída je zabalena do namespace **CodEx**, takže k ní můžete přistupovat přes notaci **CodEx.Reader**. Případně můžete na začátku svého programu rozbalit celý namespace CodEx a přistupovat ke třídě přímo:

```
using System;
using System.IO;
using CodEx;
```

Abyste mohli číst ze souboru, musíte nejprve vytvořit objekt třídy **Reader** a jako parametr konstruktoru předat cestu k souboru.

```
// Otevře soubor.
Reader inputFile = new Reader("soubor.in");

// Přečte ze souboru znak.
char ch = inputFile.Char();
```

Chcete-li pracovat s konzolí, můžete k objektu konzole přistupovat přes statickou proceduru **Reader.Console()**:

```
// Přečte z konzole znak.
char ch = Reader.Console().Char();
```

Soubor se zavře, pokud je objekt **Reader** zničen. Vzhledem k tomu, že dle specifikace jazyka C# může být finalizer zavolán kdykoli, je silně doporučeno používat k zavření souboru metodu **Close()**.

```
// Otevřeme soubor.
Reader inputFile = new Reader("soubor.in");

// Tady budeme číst ze souboru...

// A nečekáme na finalizer - zavřeme soubor sami.
inputFile.Close();
```

Reader také implementuje interface **IDisposable**, což znamená, že na přečtení souboru můžeme použít i následující konstrukci:

```
using (Reader inputFile = new Reader("soubor.in")) {
        // Tady budeme číst ze souboru...
}
// A na konci se nám objekt sám zničí (takže nemusíme volat Close()).
```

Po uzavření souboru už z něj není možné číst. Stejně tak není možné ve vstupu seekovat nebo se vrátit na začátek. Pokud z nějakého důvodu potřebujete číst vstup znovu, musíte si soubor znovu otevřít vytvořením další instance třídy **Reader**.

B.2 Přehled důležitých metod

- **`bool Char(out char c);`** — Pokusí se přečíst jeden znak ze vstupu. Pokud se čtení povedlo, je znak uložen do proměnné `c` a metoda vrací `true`. Pokud došel vstup (EOF), vrací `false`.
- **`char Char();`** — Pokusí se přečíst jeden znak ze vstupu a vrátí jej. Pokud se čtení nepovedlo nebo došel vstup, hodí výjimku IOException.
- **`bool Int(out int i);`** — Pokusí se přečíst celé číslo zapsané dekadicky ze vstupu. Pokud se čtení povedlo, je číslo uloženo do proměnné `i` a metoda vrací `true`. Pokud došel vstup (EOF), vrací `false`. V případě chyby čtení (např. proto, že na vstupu není číslo v platném formátu) hodí výjimku IOException.
- **`int Int();`** — Pokusí se přečíst celé číslo zapsané dekadicky ze vstupu a vrátí jej. Pokud se čtení z jakéhokoli důvodu nepovedlo, hodí výjimku IOException.
- **`bool Double(out double d);`** — Pokusí se přečíst reálné číslo zapsané v desetinné notaci nebo exp. tvaru ze vstupu. Pokud se čtení povedlo, je číslo uloženo do proměnné `d` a metoda vrací `true`. Pokud došel vstup (EOF), vrací `false`. V případě chyby čtení (např. proto, že na vstupu není číslo v platném formátu), hodí výjimku IOException.
- **`double Double();`** — Pokusí se přečíst reálné číslo zapsané v desetinné notaci, nebo exp. tvaru ze vstupu a vrátí jej. Pokud se čtení z jakéhokoli důvodu nepovedlo, hodí výjimku IOException.
- **`string Word();`** — Přečte jeden token (řetězec bez bílých znaků oddělený z oboru stran bílými znaky nebo začátkem/koncem souboru). Token je vrácen jako řetězec. Pokud na vstupu již nejsou žádná data, vrací `null`.
- **`string Line();`** — Přečte zbytek aktuálně čteného řádku a vrátí jej jako řetězec. Pokud na vstupu již nejsou žádná data, vrací `null`.
- **`bool EOF();`** — Vrací `true`, pokud na vstupu (resp. v souboru) už nejsou žádné další znaky k přečtení.
- **`bool SeekEOF();`** — Vrací `true`, pokud do konce vstupu (resp. souboru) už zbývají pouze bílé znaky (mezery, zalomení řádku...).

B.3 Příklady použití

Jednoduše přečteme všechna celá čísla v souboru:

```
Reader inputFile = new Reader("soubor.in");
while (!inputFile.SeekEOF()) {
        int x = inputFile.Int();
        // ... zpracuj x ...
}
inputFile.Close();
```

Přečteme vstup z konzole po řádcích:

```
string row;
while ((row = Reader.Console().Line()) != null) {
        // ... zpracuj řádek row ...
}
```