Vysoká škola polytechnická Jihlava

# Team T19

### Autorský team:

* Jiří Bukovský, bukovs05@student.vspj.cz
* Jiří Kalina, kalina08@student.vspj.cz
* Jiří Karmasin, karmasin@student.vspj.cz
* Michal Makovec, makovec@student.vspj.cz

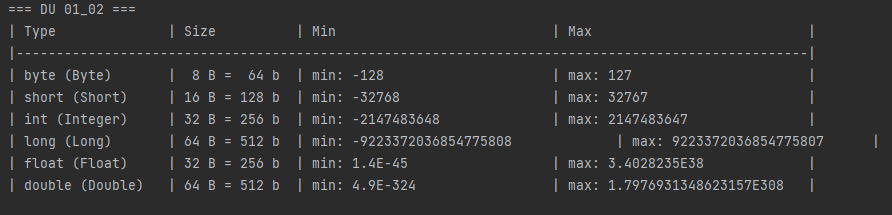
### Úkol 01\_02

Jedná se o seznámení s datovými typy a výstupy do konzole

V tomto úkolu jsem definovali tyto datové typy:

|  |
| --- |
| byte mybyte short myshort int myint long mylong float myfloat double mydouble boolean mybool char mychar |

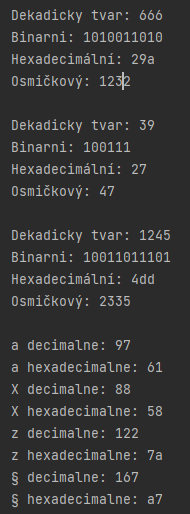
Tabulka zobrazující datové typy s jejich obalovací třídou, velikost proměnné v paměti a rozsah u číselných hodnot:



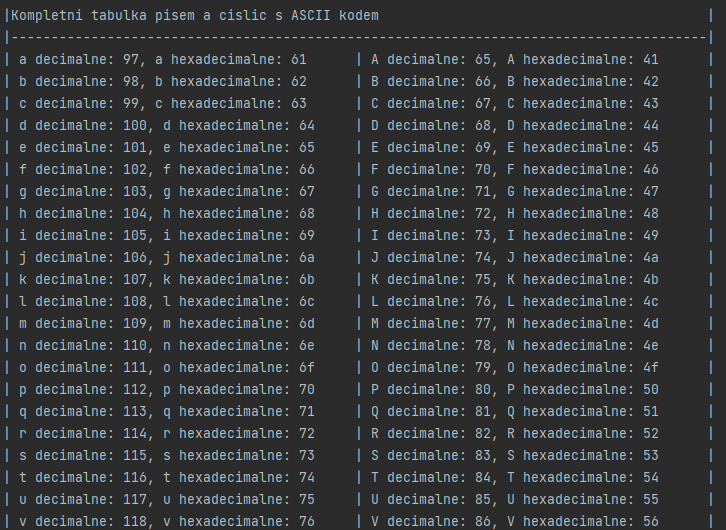
Výsledky výrazu 6 + 4 a výrazu “” + 6 + 4:



Použití metody TiskCislo(int cislo):



Část přehledové tabulky jako výstup metody kompletASCII():



### Úkol 01\_03

Konverze datových typů

Příklad volání jedné metody v hlavní funkci main:

Chybové hlášky jsou zobrazovány v jednotlivých načítacích metodách, aby bylo možné specifikovat povolený rozsah pro právě načítaný typ.

Short sh;  
 Integer i;  
 Float f;  
 Double d;  
 String s;  
 Character c;  
  
 NacteniHodnot in = new NacteniHodnot();  
  
// Short  
try{  
 System.*out*.println("Zadej short:");  
 sh = in.*nacti\_short*();  
 System.*out*.println("Short: " + sh);  
} catch (Exception e){  
 // System.out.println("ERROR: vstup");  
}

Ukázka načítací metody pro typ Integer:

Metody pro ostatní typy jsou analogické. Přímo v těchto metodách jsou vypisovány chybové zprávy při nekorektních vstupních datech.

// ========== INTEGER ========================  
public static Integer nacti\_int() throws IOException, NumberFormatException

{  
 try {  
 String s = *br*.readLine();  
 try{  
 return Integer.*valueOf*(s);  
 } catch (NumberFormatException e){  
 System.*out*.println(e.getMessage() + " Min: " + Integer.*MIN\_VALUE* + " Max: " + Integer.*MAX\_VALUE*);  
 throw e;  
 }  
 } catch (IOException e){  
 System.*out*.println(e.getMessage());  
 throw e;  
 }  
}

Výstup při správných datech:

Obsah obrázku text

Popis byl vytvořen automaticky

Výstup při špatných vstupních datech:

Obsah obrázku text

Popis byl vytvořen automaticky

Jaký je využití toho, že třída vrací proměnnou typu obalovací třídy?

### 

### Úkol 01\_04

Vytvořte program v konzolové aplikaci, který bude simulovat pohyb po šachovnici.

Směr pohybu je dán výčtovým typem:

|  |
| --- |
| enum DIRECTION{left,right,up,down} |

Šachovnice:

|  |
| --- |
| public class HraPosunSachovnici {  static final int N\_HISTORY = 255;  private byte nx;  private byte ny;  private byte x;  private byte y;   String[] history = new String[N\_HISTORY];  private int n\_tahu = 0; |

Řešení pohybu:

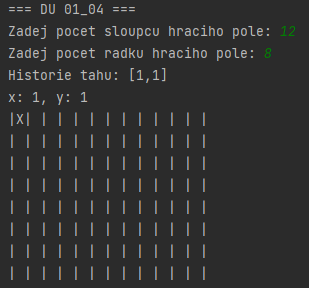
|  |
| --- |
| public boolean move(DIRECTION dir){  if(can\_move(dir)){  switch(dir) {  case up:  setY((byte) (this.getY()-1));  break;  case down:  setY((byte) (this.getY()+1));  break;  case left:  setX((byte) (this.getX()-1));  break;  case right:  setX((byte) (this.getX()+1));  }  return true;  }else{  return false;  }  } |

Zjištění zda se mohu směrem pohybovat

|  |
| --- |
| public boolean can\_move(DIRECTION dir){  switch(dir) {  case up:  return can\_up();  case down:  return can\_down();  case left:  return can\_left();  case right:  return can\_right();  }  return false;  }   private boolean can\_left(){  if(this.getX()==1) { return false; }  return true;  }  private boolean can\_right(){  if(this.getX()==this.getNX()) { return false; }  return true;  }  private boolean can\_up(){  if(this.getY()==1) { return false; }  return true;  }  private boolean can\_down(){  if(this.getY()==this.getNY()) { return false; }  return true;  } |

Výpis hry:

Ovládání hry: po spuštění je třeba zadat rozměry hracího pole a poté je možné se pohybovat pomocí kláves l, r , u, d (každý vstup zvlášť odentrovat). Konec aplikace je po zadání klávesy k.



### 

### Úkol 02\_U1

Bod ve 2D prostoru

Základní určení souřadnic bodu pomocí třídy a konstruktor

|  |
| --- |
| *// Bod2D.java* public class Bod2D {  private double x;  private double y;    public Bod2D(){  x = 0;  y = 0;  }   public Bod2D(double n\_x, double n\_y){  x = n\_x;  y = n\_y;  } |

Barva RGB pomocí primitivního datového typu **byte**. Byte nabývá hodnot od -128 do +127. RGB hodnoty jsou v rozmezí 0 až 255. Rozmezí je ošetřené výjimkou, aby nebylo možno zadat jiný barevný rozsah. Na vstupu získáváme “integer” a odečítáme od něj -128 abychom dostali povolenou hodnotu pro “byte”. Na výstupu provádíme opačnou operaci, abychom z “byte” získali “integer”.

|  |
| --- |
| *// BarvaRGB.java* public class BarvaRGB {  byte r; *// -128 .. 127*  byte g;  byte b;  float intens;   public BarvaRGB(){  r = -128;  g = -128;  b = -128;  intens = (float)0.0;  }  public BarvaRGB(int n\_r, int n\_g, int n\_b) throws Exception{  if (n\_r<0 || n\_r >255 || n\_g<0 || n\_g >255 || n\_b<0 || n\_b >255) {  throw new Exception("ERROR: Out of color range 0 .. 255");  }  r = (byte)(n\_r-128);  g = (byte)(n\_g-128);  b = (byte)(n\_b-128);  intens = (float)(r +g + b)/3;  } |

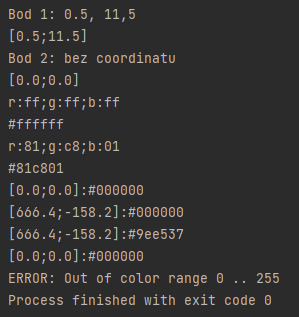
Třída Bod2D\_Extend rozšiřující existující třídu Bod2D. Jako atribut je přidaný objekt typu BarvaRGB. V konstruktoru jsou volány bezparametrové kontruktory obou tříd - jak třídy předka, tak třídy atributu barva.

|  |
| --- |
| public class Bod2D\_Extend extends Bod2D{  BarvaRGB barva;   public Bod2D\_Extend(){  super();  barva = new BarvaRGB();  }   public Bod2D\_Extend(double n\_x, double n\_y){  super(n\_x,n\_y);  barva = new BarvaRGB();  } |

Výpis hodnoty barvy v RGB a Hexadecimálním formátu - fragment kódu

|  |
| --- |
| *// BarvaRGBb.java* public class BarvaRGBb {  byte r; *// -128 .. 127*  byte g;  byte b;  float intens;   public BarvaRGBb(){  r = -128;  g = -128;  b = -128;  intens = (float)0.0;  }  public BarvaRGBb(int n\_r, int n\_g, int n\_b) throws Exception{  if (n\_r<0 || n\_r >255 || n\_g<0 || n\_g >255 || n\_b<0 || n\_b >255) {  throw new Exception("Out of color range 0 .. 255");  }  r = (byte)(n\_r-128);  g = (byte)(n\_g-128);  b = (byte)(n\_b-128);  intens = (float)(r +g + b)/3;  } *// . . .* public void print(){  String r\_rhx = Integer.toHexString((int)r+128);  String g\_rhx = Integer.toHexString((int)g+128);  String b\_rhx = Integer.toHexString((int)b+128);   System.out.println("r:"+r\_rhx+";g:"+g\_rhx+";b:"+b\_rhx);  }  public void print2(){  String r\_rhx = Integer.toHexString((int)r+128);  String g\_rhx = Integer.toHexString((int)g+128);  String b\_rhx = Integer.toHexString((int)b+128); *// . . .* } |

Výstup z konzole po spuštění programu



### 

### Úkol 02\_U2

Objekt "nádrže"

Hodnoty Type jsme udělali jako výčtový typ specifikující kapalinu v nádrži.

|  |
| --- |
| *// Nádrž.java* public class Nadrz {   int max;  int state;  enum Type{BENZIN, NAFTA, ETHANOL, WASSER};  Type type;   public Nadrz(){  max = 0;  state = 0;  type = Type.WASSER;  }  *// Následují metody gettry a settry* *// . . .* |

Důležité metody:

|  |
| --- |
| *// Nádrž.java*  public void print(){   String out = getState() + "/" + getMax() + "("+ naplneni()\*100 +"%),"+getType();  System.out.println(out);  }    public void pridat(int x) throws MyException\_PlnaNadrz{  if((x+state) > max) throw new MyException\_PlnaNadrz("Nadrz je prilis plna pro pridani");  else{  setState(getState()+x);  }  }   public void odebrat(int x) throws MyException\_PrazdnaNadrz{  if((state - x) < 0) throw new MyException\_PrazdnaNadrz("Nadrz je moc prazdna na odebrani");  else{  setState(getState()-x);  } |

Vlastní třídy pro exception:

|  |
| --- |
| *// MyException\_PlnaNadrz.java* public class MyException\_PlnaNadrz extends Exception{   public MyException\_PlnaNadrz(String em){  super(em);  } } *// MyException\_PrazdnaNadrz.java* public class MyException\_PrazdnaNadrz extends Exception{  public MyException\_PrazdnaNadrz(String em){  super(em);  }  } |

Součástí zdrojového kódu je hlavní funkce napsaná tak, aby se daly všechny mezní stavy naplnění a odčerpání z nádrže ošetřit.

Console.out:

