В ТЕМАТА:

- ✓ Как се реализира модел за решаване на задачи, базиран на целочислени типове данни.
- ✓ Как се прилагат и анализират резултатите от основните аритметични операции.

ЗАДАЧА 1

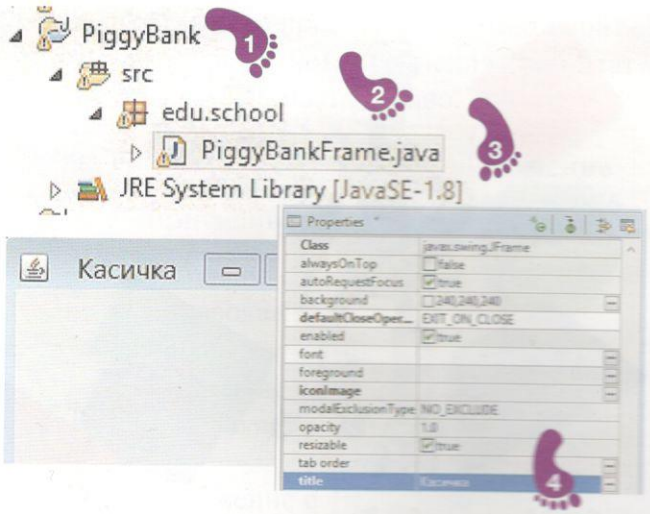
Реализирайте приложение **PiggyBank**, което дава възможност да се въведе брой на монетите от всеки вид в касичка и пресмята общата сума в лева и стотинки.

ПРОЕКТИРАНЕ НА ГПИ

1. За извеждане на резултата е удобно да се използват етикети.
2. За въвеждане на броя на монетите от всеки вид – по едно текстово поле.
3. За задаване на функционалност – бутони: за пресмятане; за изчистване на съдържанието; за изход.

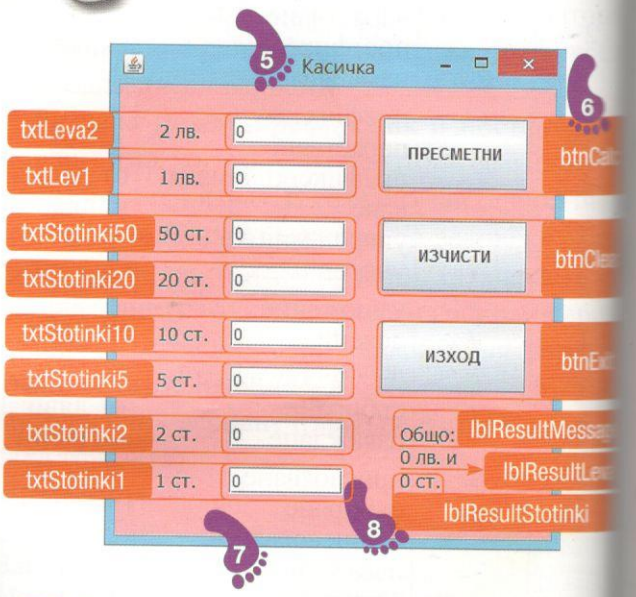
СТЪПКА ПО СТЪПКА – *създаване на форма*

1. Създайте ново приложение **PiggyBank**.
2. В папката **src** добавете пакет ***edu.school***.
3. В пакета създайте нов *JFrame* и го именувайте **PiggyBankFrame**.
4. Задайте заглавие (*title*) **Касичка**.



СТЪПКА ПО СТЪПКА – *контроли*

5. Разположете контролите върху формата.
6. Преименувайте всички контроли.
7. Променете текстовете на етикетите и бутоните.
8. Задайте стойност 0 в текстовите полета.



СТЪПКА ПО СТЪПКА – *бутони*

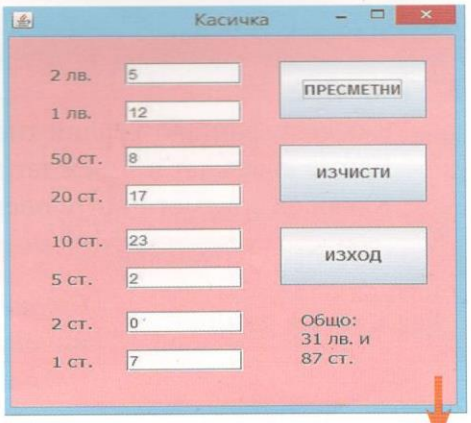
9. Добавете функционалност към бутона ***Изход***.
10. Добавете функционалност към бутона ***Изчисти***.

```
System.exit(0);
```

```
txtLeva2.setText("0");
txtLev1.setText("0");
txtStotinki50.setText("0");
txtStotinki20.setText("0");
txtStotinki10.setText("0");
txtStotinki5.setText("0");
txtStotinki2.setText("0");
txtStotinka1.setText("0");
lblResultLeva.setText("0 лв. и");
lblResultStotinki.setText("0 ст.");
```

СТЪПКА ПО СТЪПКА – *бутон Пресметни*

11. Дефинирайте целочислени променливи, в които да запишете входните данни.
12. Пресметнете общата сума в стотинки.
13. Намерете колко лева се съдържат в общата сума.
14. Извадете от общата сума левовете и представете остатъка в стотинки.
15. Изведете резултата в съответните етикети.




```
int leva2 = Integer.parseInt(txtLeva2.getText());
int lev1 = Integer.parseInt(txtLev1.getText());
int stotinki50 = Integer.parseInt(txtStotinki50.getText());
int stotinki20 = Integer.parseInt(txtStotinki20.getText());
int stotinki10 = Integer.parseInt(txtStotinki10.getText());
int stotinki5 = Integer.parseInt(txtStotinki5.getText());
int stotinki2 = Integer.parseInt(txtStotinki2.getText());
int stotinka1 = Integer.parseInt(txtStotinka1.getText());
```

11

```
leva2 = 5
lev1 = 12
stotinki50 = 8
stotinki20 = 17
stotinki10 = 23
stotinki5 = 2
stotinki2 = 0
stotinka1 = 7
```

```
int totalStotinki = 200 * leva2 + 100 * lev1 +
                    50 * stotinki50 + 20 * stotinki20 +
                    10 * stotinki10 + 5 * stotinki5 +
                    2 * stotinki2 + stotinka1;
```

12

```
totalStotinki =
    200 * 5 + 100 * 12 +
    50 * 8 + 20 * 17 +
    10 * 23 + 5 * 2 +
    2 * 0 + 7;
```

```
int resultLeva = totalStotinki / 100;
int resultStotinki = totalStotinki % 100;
```

14 13

```
totalStotinki =
    1000 + 1200 +
    400 + 340 +
    230 + 10 + 0 +  7;
```

```
totalStotinki = 3187;
```

За да намерим общата сума в стотинки, за всеки вид монети умножаваме броя им по тяхната равностойност в стотинки и събираме получените резултати.

Един лев съдържа 100 стотинки. Следователно броят на левовете е цялата част, която се получава, когато разделим общия брой стотинки на 100.

Останалите стотинки от общата сума може да намерим, като пресметнем остатъка при целочислено деление на 100.

```
String resultLevaMessage =
        Integer.toString(resultLeva) + " лв. и ";
String resultStotinkiMessage =
        Integer.toString(resultStotinki) + " ст.";
```

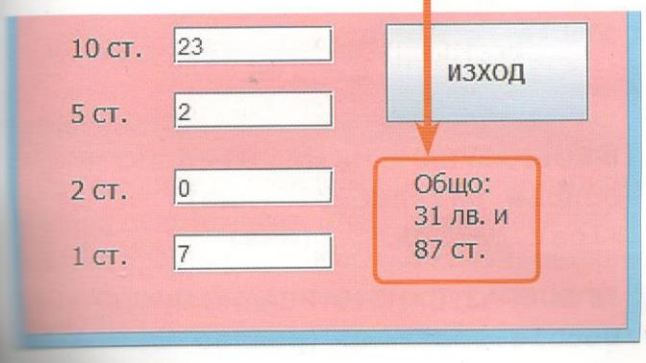
```
resultLeva = 3187 / 100;
resultStotinki = 3187 % 100;
```

```
lblResultLeva.setText(resultLevaMessage);
lblResultStotinki.setText(resultStotinkiMessage);
```

15

```
resultLeva = 31;
resultStotinki = 87;
```

```
resultLevaMessage = "31" + " лв. и ";
resultStotinkiMessage = "87" + "ст.";
```



ЗАДАЧА 2

Направете приложение **EggsPack**, което да помогне на RxC001 да определи колко кутии с яйца трябва да купи от супермаркета, ако е известно по колко яйца има в една кутия и колко яйца общо са му нужни.

3.1.2. ЦЕЛОЧИСЛЕНИ ТИПОВЕ ДАННИ

В ТЕМАТА:

- ✓ Как се дефинират целочислени типове данни.
- ✓ Какви са конвенциите в Java за именуване на константи и променливи.
- ✓ Кои са целочислените аритметични операции и какъв е техният приоритет.

ПОНЯТИЕ	ОПИСАНИЕ
Целочислен тип данни	Служи за представяне на цели числа в ограничен интервал. Основните целочислени типове в Java са `int`, `byte`, `short` и `long`.
Конвенция за именуване на константи	При именуване на константи се използват само главни букви. Използват се значещи думи, които да описват (подсказват) значението на константата. За да се отличат отделните думи в наименованието на константа, се използва долна черта (_), която да ги слепва. Евентуално в името на константа може да се използват и цифри, но при спазване на правилата за запис на идентификатори.
Конвенция за именуване на променливи	При именуване на променливи се използват значещи думи, които да описват (подсказват) значението на променливата. Първата дума в името на променливата е изписана изцяло с малки букви, а всяка следваща дума е долепена до предходната и има начална главна буква, следвана от малки букви. Може да се използват и цифри при спазване на правилата за запис на идентификатори. Обикновено не се използва символът долна черта (_). Нарича се *camelCase* (стил „камила"), защото редуването на малки и главни букви наподобява гърбиците на камила.

КОНВЕНЦИИ ЗА ИМЕНУВАНЕ НА КОНСТАНТИ И ПРОМЕНЛИВИ

Когато се именуват константи и променливи, е добре да се спазват общоприетите конвенции (правила) за именуване. Това улеснява четенето на програмен код от програмистите. Ако дадена променлива или константа е именувана в противоречие с конвенциите, това не влияе на коректността на програмата, а просто се смята за лош **стил на програмиране**.

ЦЕЛИ ЧИСЛА

Има два вида цели числа – със знак и без знак. **Целите числа без знак** са последователност от цифри. Например `65`, `0`, `5892431`. Често допускана грешка е поставянето на интервали между цифрите. Например `1 000 000` е некоректен запис и трябва да се запише като `1000000`. **Целите числа със знак** са положителни или отрицателни в зависимост от използвания знак (+) или (–), поставен пред цялото число.

ПРИМЕРИ

`+5`, `+67032` са положителни цели числа със знак, а `-12`, `-9427` са отрицателни числа.

ЗАДАЧИ 1

- Според конвенциите за именуване кои от следните идентификатори са имена на променливи и кои – на константи?
  `X`, `y`, `min`, `MAX`, `myName`, `U2`
- Кои цели числа са изписани правилно?
  `+0`,`-0`,`1 300`,`13`,`2017`,`+1`,`-125000`

ЦЕЛОЧИСЛЕНИ ТИПОВЕ ДАННИ

Тип	Памет	Множество от стойности	Стойност по подразбиране
`int`	4 байта	Min: -2^{31} Max: $2^{31}-1$	0
`byte`	1 байт	Min: -2^{7} Max: $2^{7}-1$	0
`short`	2 байта	Min: -2^{15} Max: $2^{15}-1$	0
`long`	8 байта	Min: -2^{63} Max: $2^{63}-1$	0L

ЦЕЛОЧИСЛЕНИ КОНСТАНТИ

ПРИМЕРИ

```
final byte SECONDS_PER_MINUTE = 60;
final short SECONDS_PER_HOUR = 3600;
final int SECONDS_PER_DAY = 86400;
final int SECONDS_PER_YEAR = 31536000;
final long SECONDS_PER_CENTURY =
           3153600000;
```

Обикновено като именувани константи се задават стойности, които остават непроменени по време на изпълнение на програмата. Понякога обаче се налага актуализиране на константите от програмиста, дори и промяна на техния тип.

ЗАДАЧИ 2

- Дефинирайте именувана константа за броя на държавите членки на Европейския съюз. Потърсете в интернет информация колко пъти се е променял броят им. Каква стойност ще присвоите на константата?
- В дадения пример за целочислени константи не е отчетен броят на високосните години в един век. От друга страна, за да се доближават максимално слънчевото време и гражданското време (отчитано по часовник), се правят и допълнителни корекции. Потърсете информация за т. нар. високосна секунда и коригирайте стойността на константата `SECONDS_PER_CENTURY`.

ЦЕЛОЧИСЛЕНИ ПРОМЕНЛИВИ

ПРИМЕРИ – *декларация*

```
byte decimalValue;
short totalNumberOfDays;
int myList, listSize, indexOfMax;
long low, high, city1, city2;
```

ПРОМЕНЛИВИ — ИНИЦИАЛИЗАЦИЯ

```
тип име = стойност ;
тип име = стойност ,
     име = стойност ;
```

ПРИМЕРИ – *инициализация*

```
byte decimalValue = -20;
short totalNumberOfDays = 31;
int listSize = 104, indexOfMax = 95;
long totalSum = 438478320,
     low = -548725, high = 3428922;
```

ВГРАДЕНИ ФУНКЦИИ ЗА ПРЕОБРАЗУВАНЕ

ПРЕОБРАЗУВАНЕ НА НИЗ В ЦЯЛО ЧИСЛО

```
Integer.parseInt(низ)
Byte.parseByte(низ)
Short.parseShort(низ)
Long.parseLong(низ)
```

ПРЕОБРАЗУВАНЕ НА ЦЯЛО ЧИСЛО В НИЗ

```
Integer.toString(цяло число)
Byte.toString(цяло число)
Short.toString(цяло число)
Long.toString(цяло число)
```

ПРИМЕРИ

```
Integer.parseInt("-1234")
```

Резултат: *цялото число* –1234

```
Integer.parseInt("1.5")
```

Резултат: *грешка*

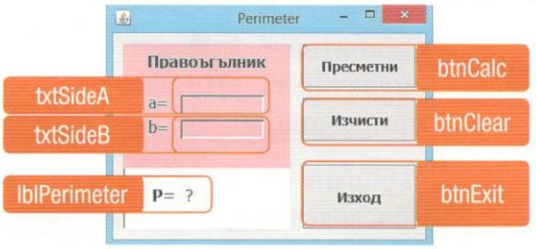
```
Integer.toString(586)
```

Резултат: *низ* "586"

ВЪВЕЖДАНЕ И ИЗВЕЖДАНЕ НА ДАННИ ОТ ЦЕЛОЧИСЛЕН ТИП

ЗАДАЧА 3

- Направете приложение, което пресмята периметъра на правоъгълник, ако дължините на страните му са зададени като целочислени стойности в сантиметри.



Често за въвеждане на целочислени данни се използват текстови полета, в които данните се въвеждат като низове и се налага преобразуването им в целочислена стойност.

ПРИМЕРИ

```
int sideA = Integer.parseInt(txtSideA.getText());
int sideB = Integer.parseInt(txtSideB.getText());
```

Обикновено целочислените стойности се извеждат в етикети. Това налага преобразуването им в низ.

ПРИМЕР

```
lblPerimeter.setText(Integer.toString(perimeter));
```

ОСНОВНИ АРИТМЕТИЧНИ ОПЕРАЦИИ

Оператор	Име	Пример	Резултат
+	събиране	35 + 2	37
-	изваждане	35 – 2	33
*	умножение	35 * 2	70
/	целочислено деление	35 / 2	17
%	деление по модул (остатък при целочислено деление)	35 % 2	1

За аритметичните операции в Java са в сила същите свойства, както и в математиката – разместително, съдружително и разпределително.

ДРУГИ ОПЕРАТОРИ ЗА АРИТМЕТИЧНИ ОПЕРАЦИИ

Както при низовете има съкратен оператор за присвояване, така и при аритметичните операции има по един такъв оператор за всяка основна аритметична операция. При тях задължително променливата, която стои отляво, трябва да е инициализирана. Първо се пресмята аритметичният израз от дясната страна. След това върху получения резултат и стойността на променливата отляво се прилага съответната аритметична операция и новополученият резултат се записва в променливата отляво.

Оператор	Съкратен оператор за присвояване	Пример	Значение при $x = 35$
+=	събиране с присвояване	x += 2;	$x = x + 2$; $x = 35 + 2$; x е 37.
-=	изваждане с присвояване	x –= 2;	$x = x - 2$; $x = 35 - 2$; x е 33.
*=	умножение с присвояване	x *= 2;	$x = x * 2$; $x = 35 * 2$; x е 70.
/=	деление с присвояване	x /= 2;	$x = x/2$; $x = 35/2$; x е 17.
%=	деление по модул с присвояване	x %= 2;	$x = x \% 2$; $x = 35 \% 2$; x е 1.

Оператор	Име	Пример	Значение
++var	префиксно нарастване	++x	$x = x + 1$; x е 36.
var++	постфиксно нарастване	x ++	$x = x + 1$; x е 36.
--var	префиксно намаляване	--x	$x = x - 1$; x е 34.
var--	постфиксно намаляване	x --	$x = x - 1$; x е 34.

Често в цикличните алгоритми се налага да се използват броячи, като на всяка стъпка стойността на брояча се увеличава/намалява с единица. В тези случаи е по-удобно да се използват оператори за префиксно (постфиксно) нарастване (намаляване). По принцип, когато се използват като самостоятелни оператори, няма значение дали ще се използва префиксен, или постфиксен вариант.

Разлика между префиксния и постфиксния вариант има, когато тези оператори се използват в аритметичен израз. В този случай оценката на аритметичния израз може да се разглежда като действие в две отделни стъпки: (а) нарастване (намаляване) и (б) изпълнение на основни аритметични операции. При префиксния вариант първо се изпълнява (а), а след това – (б). При постфиксния вариант първо се изпълнява (б), а след това – (а).

ПРИМЕРИ

```
int count = 4;
int number = count++ + 2;
```
Изпълнение: (б) `number = count + 2;`
(а) `count++;`
Резултат: number е 6, count е 5.

```
int count = 4;
int number = ++count + 2;
```
Изпълнение: (а) `++count;`
(б) `number = count + 2;`
Резултат: count е 5, number е 7.

```
int count = 4;
int number = --count + 2;
```
Изпълнение: (а) `--count;`
(б) `number = count + 2;`
Резултат: count е 3, number е 5.

```
int count = 4;
int number = count-- + 2;
```
Изпълнение: (б) `number = count + 2;`
(а) `count--;`
Резултат: number е 6, count 3.

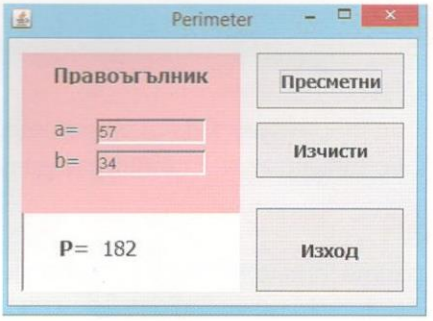
ПРИОРИТЕТ НА АРИТМЕТИЧНИТЕ ОПЕРАЦИИ

Аритметичните оператори може да бъдат разделени на: **унарни**, **бинарни** и **оператори за съкратено присвояване**. Унарните оператори се прилагат върху един аргумент (една стойност). Такъв е унарният минус, който служи за промяна на знака на израз. Други унарни оператори са за префиксно нарастване и намаляване. Основните аритметични операции са бинарни, т.е. прилагат се върху два аргумента.

Приоритет*	Оператор	Описание	Асоциативност
1	( )	скоби	от ляво надясно
2	++ --	постфиксно нарастване постфиксно намаляване	от дясно наляво
3	++ -- - +	префиксно нарастване префиксно намаляване унарен минус унарен плюс	от дясно наляво
4	* / %	умножение деление деление по модул	от ляво надясно
5	+ -	събиране изваждане	от ляво надясно
6	= += -= *= /= %=	присвояване събиране с присвояване изваждане с присвояване умножение с присвояване деление с присвояване деление по модул с присвояване	от дясно наляво

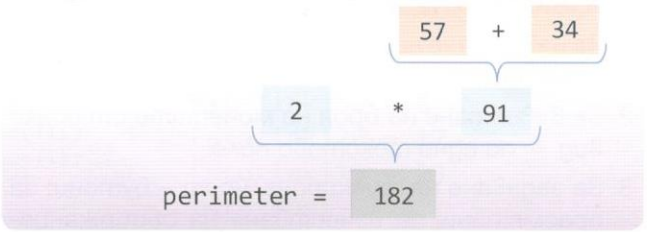
(*) По-малките числа означават по-висок приоритет.

Приоритетът на операциите може да се промени с кръгли **скоби** ( ). Както при математическите изрази, и тук може да се използват вложени един в друг изрази, заградени със скоби. Изпълнението на операциите започва от вътре навън. Честа грешка е да се пропусне да се затвори някоя скоба или да се използват други видове скоби – квадратни или фигурни.



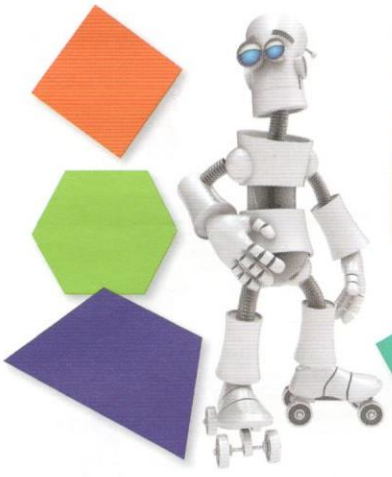
ПРИМЕРИ

```
int perimeter = 2 * (sideA + sideB);
```

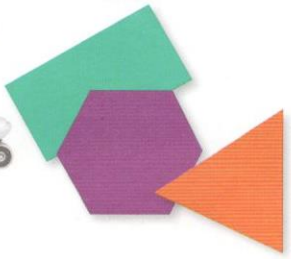


Трик!
num%10 връща последната цифра на числото **num**.

Трик!
num/10 връща числото **num** без последната му цифра.



Трик!
num%2 връща нула, ако числото **num** е четно.



ЗАДАЧА 4

Преобразувайте приложението, което пресмята периметъра на правоъгълник, така, че да пресмята периметър на триъгълник, правилен многоъгълник, квадрат и трапец. Страните на всички геометрични фигури са зададени като цели числа в сантиметри.