Pattern Matching

И его воображаемые друзья

Тагир Валеев

Disclaimer

Ничего этого нет, это всё плод воображения!

2016

Ноябрь: Java Language and Platform Futures: A Sneak Peek by Brian Goetz (Devoxx Belgium)

2017

Март: первое обсуждение в amber-dev
Май: JEP draft: Pattern Matching
Июнь: Submitted as JEP 305

2018

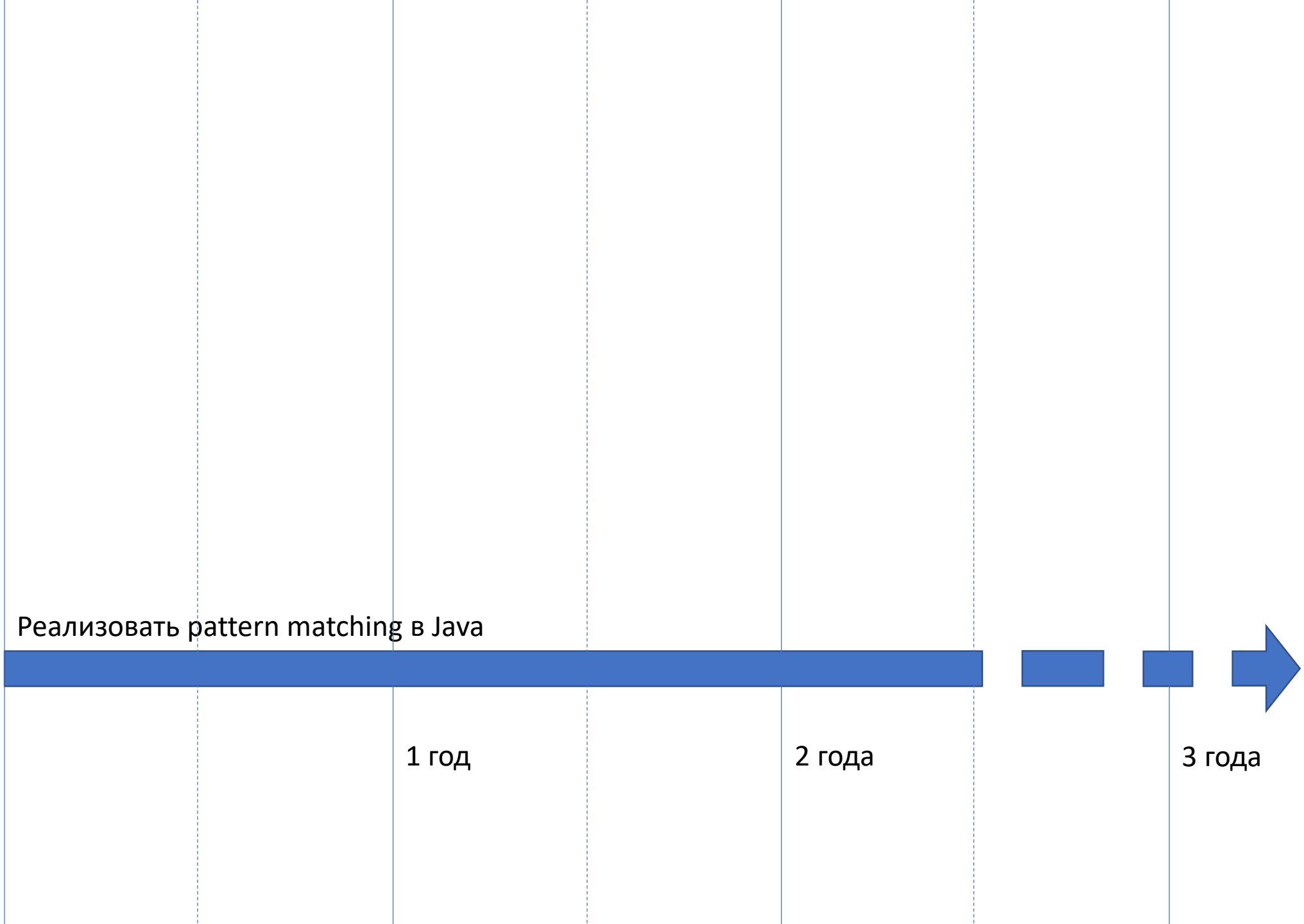
Сентябрь: обновлённые документы: Pattern Matching и Pattern Matching Semantics

2019

Март: Java 12

Сентябрь: Java 13

2020


Реализовать pattern matching в Java
1 год
2 года
3 года

Выносить ребёнка

Построить Дредноут

Долететь от Земли до Юпитера

Реализовать pattern matching в Java

1 год

2 года

3 года

Паттерн матчинг:

Есть паттерны, которые что-то матчат (или нет)

Какие бывают паттерны?

(spoiler: разные)

- ✓ Any pattern:
  - _

Какие бывают паттерны?

(spoiler: разные)

- ✓ Any pattern:
  - _
- ✓ Constant pattern:
  - 1
  - true
  - TimeUnit.SECONDS
  - null

Какие бывают паттерны?

(spoiler: разные)

- ✓ Any pattern:
  - _
- ✓ Constant pattern:
  - 1
  - true
  - TimeUnit.SECONDS
  - null
- ✓ Type pattern:
  - String s

Какие бывают паттерны?

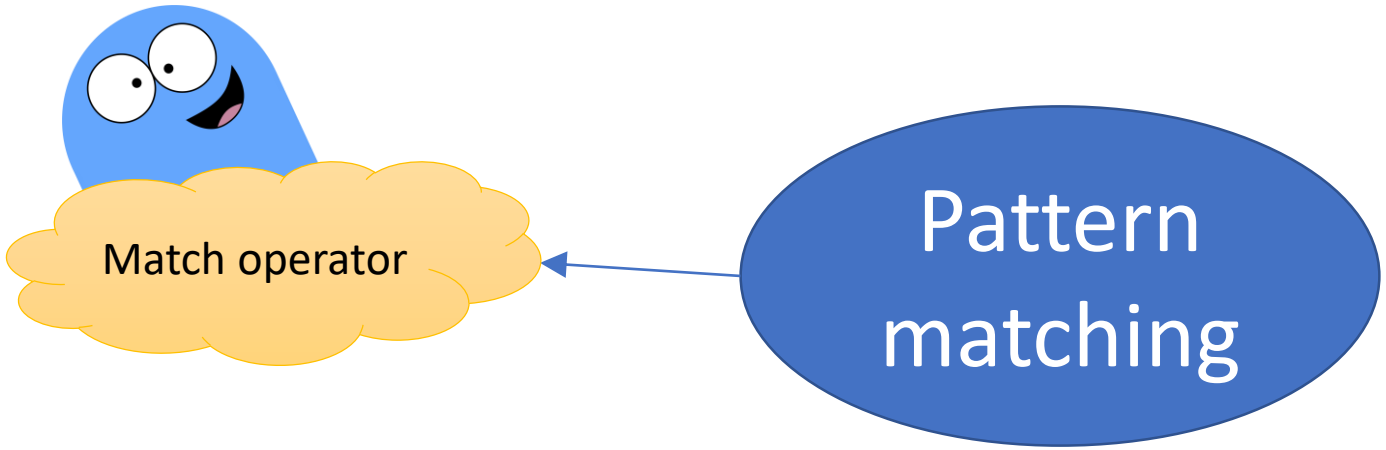
(spoiler: разные)

- ✓ Any pattern:
  - _
- ✓ Constant pattern:
  - 1
  - true
  - TimeUnit.SECONDS
  - null
- ✓ Type pattern:
  - String s
- ✓ Var pattern:
  - var x

Какие бывают паттерны?

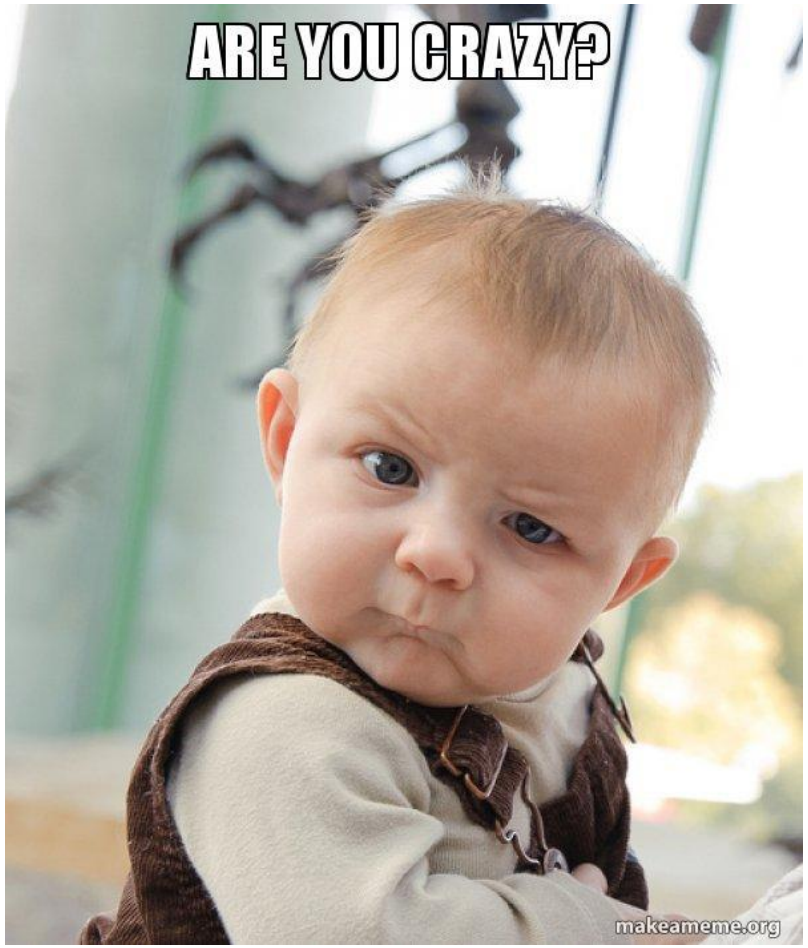
(spoiler: разные)

- ✓ Any pattern:
  - _
- ✓ Constant pattern:
  - 1
  - true
  - TimeUnit.SECONDS
  - null
- ✓ Type pattern:
  - String s
- ✓ Var pattern:
  - var x
- ✓ Deconstruction pattern:
  - Optional(String s)
  - Node(Node left, Node right)


Match operator
Pattern matching

Фи

```
void test(Object obj) {
  if (obj matches _) {
    System.out.println("Are you crazy?");
  }
}
```



Фи

```
void test(Object obj) {
  if (obj matches _) {
    System.out.println("Are you crazy?");
  }
  if (obj matches var anotherObj) {
    System.out.println("Oh well... " + anotherObj);
  }
}
```

Фи

```
void test(Object obj) {
  if (obj matches _) {
    System.out.println("Are you crazy?");
  }
  if (obj matches var anotherObj) {
    System.out.println("Oh well... " + anotherObj);
  }
  if (obj matches 42) {
    System.out.println("The Answer!");
  }
  if (obj matches "Joker") {
    System.out.println("Joker!!!");
  }
}
```

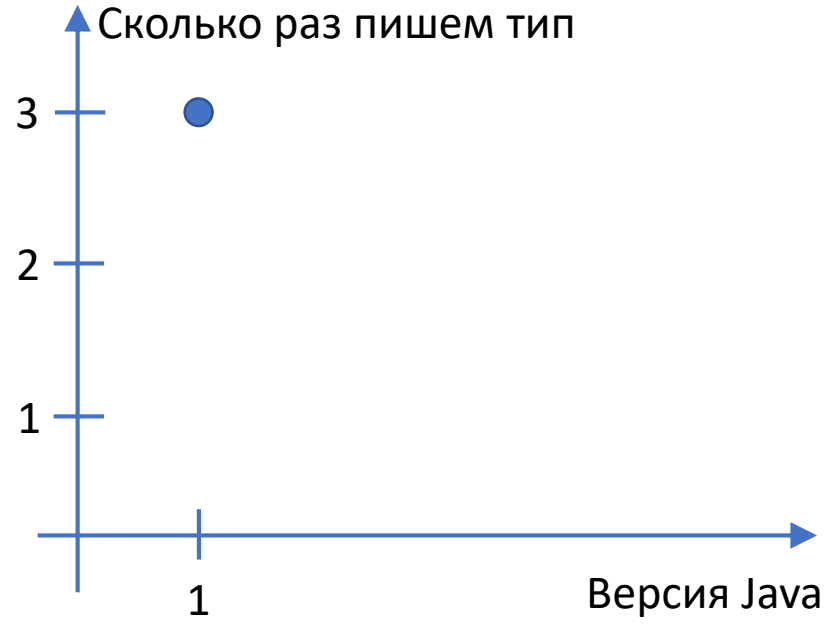
Фи

Ня

```
void test(Object obj) {
  if (obj matches _) {
    System.out.println("Are you crazy?");
  }
  if (obj matches var anotherObj) {
    System.out.println("Oh well... " + anotherObj);
  }
  if (obj matches 42) {
    System.out.println("The Answer!");
  }
  if (obj matches "Joker") {
    System.out.println("Joker!!!");
  }
  if (obj matches Number n) {
    System.out.println(n.longValue());
  }
  if (obj matches Optional(Number n)) {
    System.out.println("Wrapped "+n.longValue());
  }
}
```

Java 1:

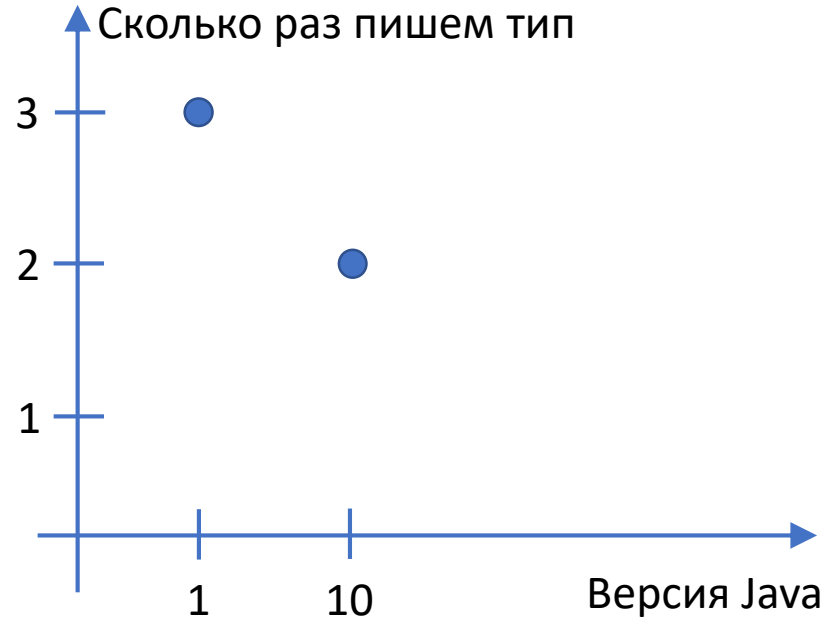
```
void test(Object obj) {
  if (obj instanceof Number) {
    Number n = (Number) obj;
    System.out.println(n.longValue());
  }
}
```



Java 1:

Java 10:

```
void test(Object obj) {
  if (obj instanceof Number) {
    Number n = (Number) obj;
    System.out.println(n.longValue());
  }

  if (obj instanceof Number) {
    var n = (Number) obj;
    System.out.println(n.longValue());
  }

}
```

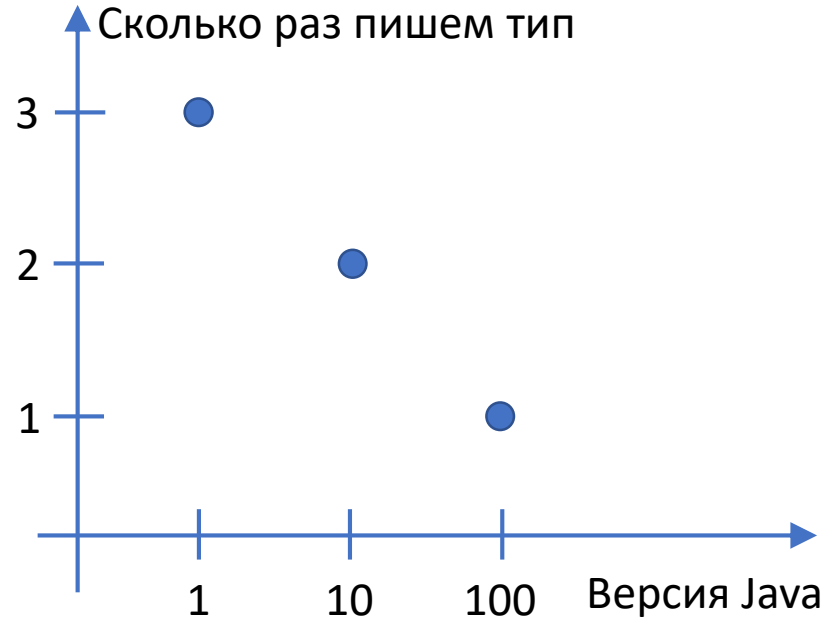


Java 1:

Java 10:

Java 100:

```
void test(Object obj) {
  if (obj instanceof Number) {
    Number n = (Number) obj;
    System.out.println(n.longValue());
  }

  if (obj instanceof Number) {
    var n = (Number) obj;
    System.out.println(n.longValue());
  }

  if (obj matches Number n) {
    System.out.println(n.longValue());
  }
}
```



```
void test(Object obj) {
  if (obj instanceof Number) {
    Number n = (Number) obj;
    System.out.println(n.longValue());
  }

  if (obj instanceof Number) {
    var n = (Number) obj;
    System.out.println(n.longValue());
  }

  if (obj instanceof Number n) {
    System.out.println(n.longValue());
  }
}
```

<expression> **instanceof** <type>

<expression> **instanceof** <pattern>

<expression> **instanceof** <type>

<expression> **instanceof** <pattern>

```
void test(Object obj) {
  if (obj instanceof String) {



  }
}
```

<expression> **instanceof** <type>

<expression> **instanceof** <pattern>

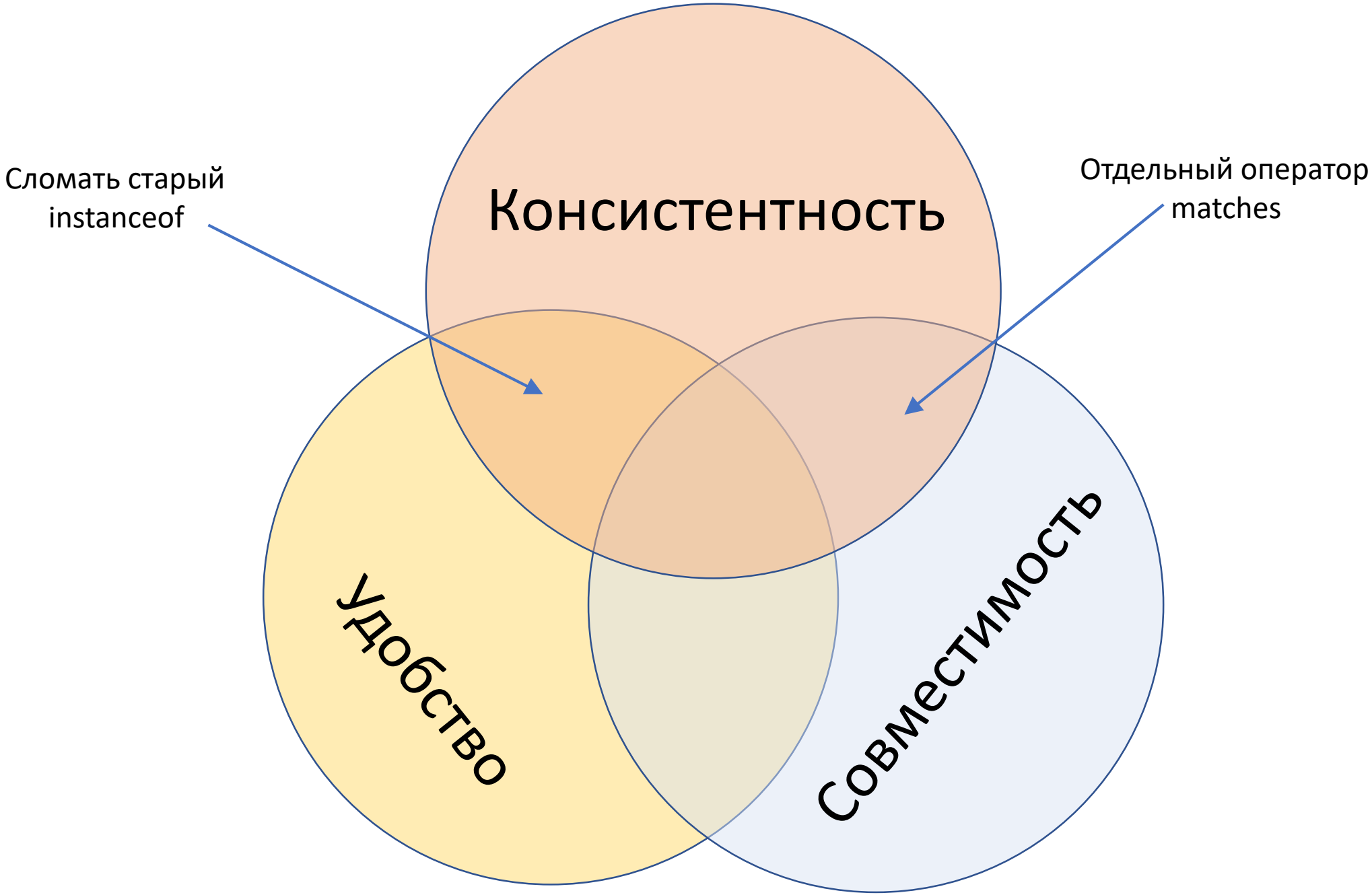
```
static final int String = 1;

void test(Object obj) {
  if (obj instanceof String) {
    
  }
}
```

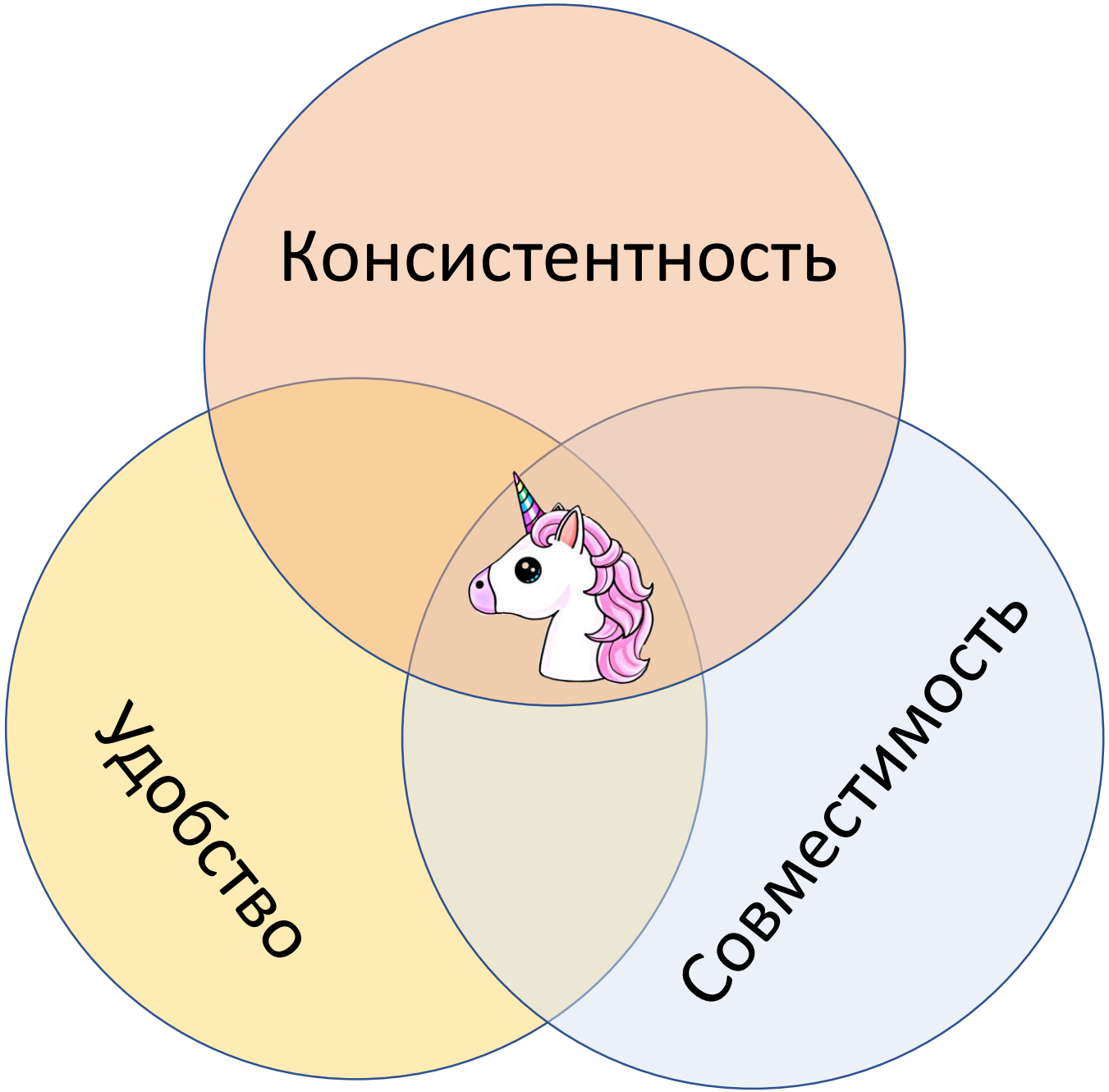
Problem, language designers?

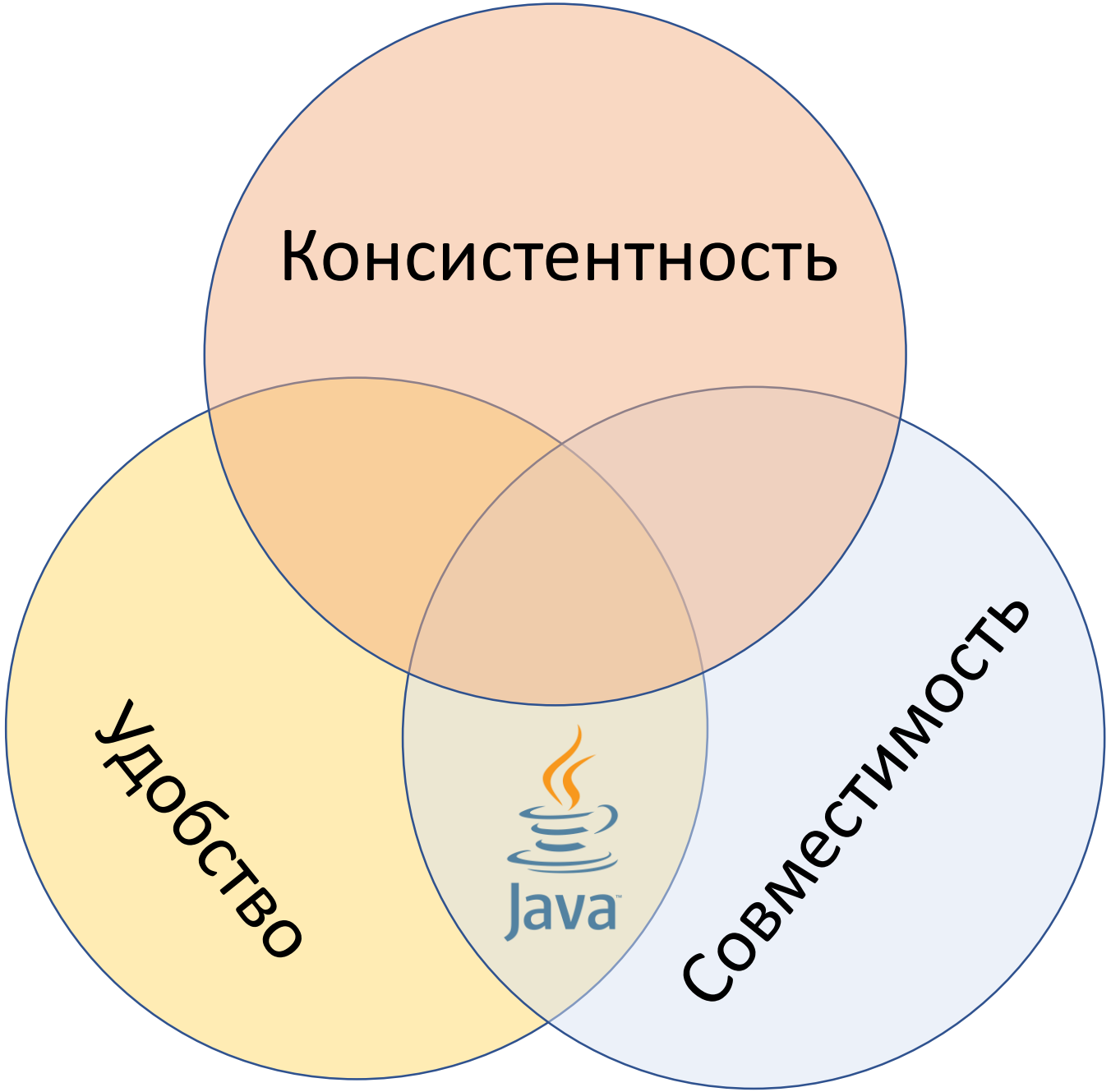
<expression> **instanceof** <type>

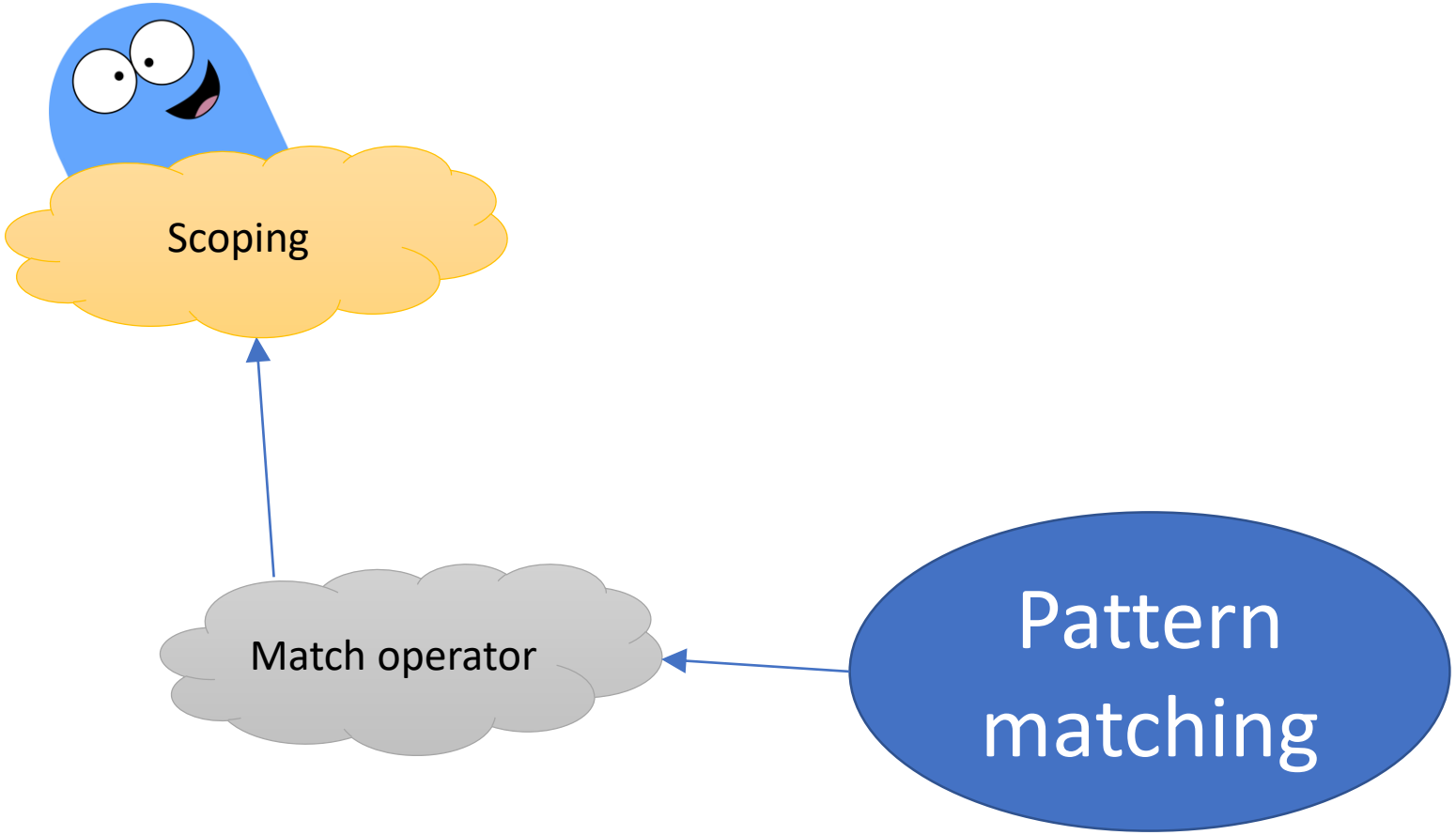
<expression> **instanceof** <type pattern>

<expression> **instanceof** <deconstruction pattern>


Сломать старый instanceof
Отдельный оператор matches
Консистентность
удобство
Совместимость


Консистентность
Удобство
Совместимость


Консистентность
удобство
Совместимость
Java


Scoping
Match operator
Pattern matching

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}
```

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}
```

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}
```

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}
```

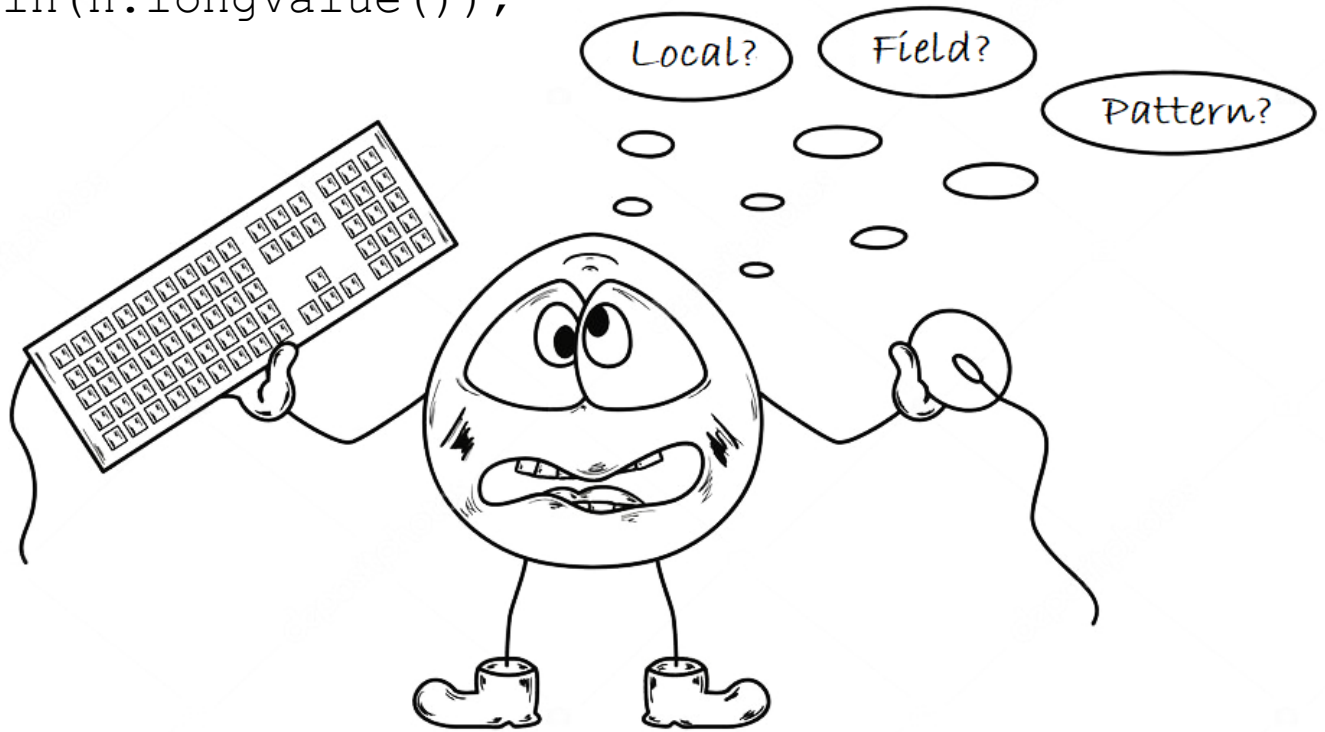
```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {}
```

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {
  System.out.println("Zero or not a number at all");
} else {
  System.out.println(n.longValue());
}
```

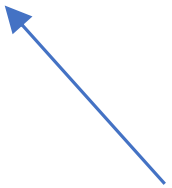
```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {
  System.out.println("Zero or not a number at all");
} else {
  System.out.println(n.longValue());
}
```

```java
if(obj instanceof Number n) {
  System.out.println(n.longValue());
}

if(obj instanceof Number n && n.longValue() == 0) {}

if(obj instanceof Number n || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {}

if(!(obj instanceof Number n) || n.longValue() == 0) {
  System.out.println("Zero or not a number at all");
} else {
  System.out.println(n.longValue());
}

if(!(obj instanceof Number n) || n.longValue() == 0) {
  throw new RuntimeException("Zero or not a number at all");
}
System.out.println(n.longValue());
```

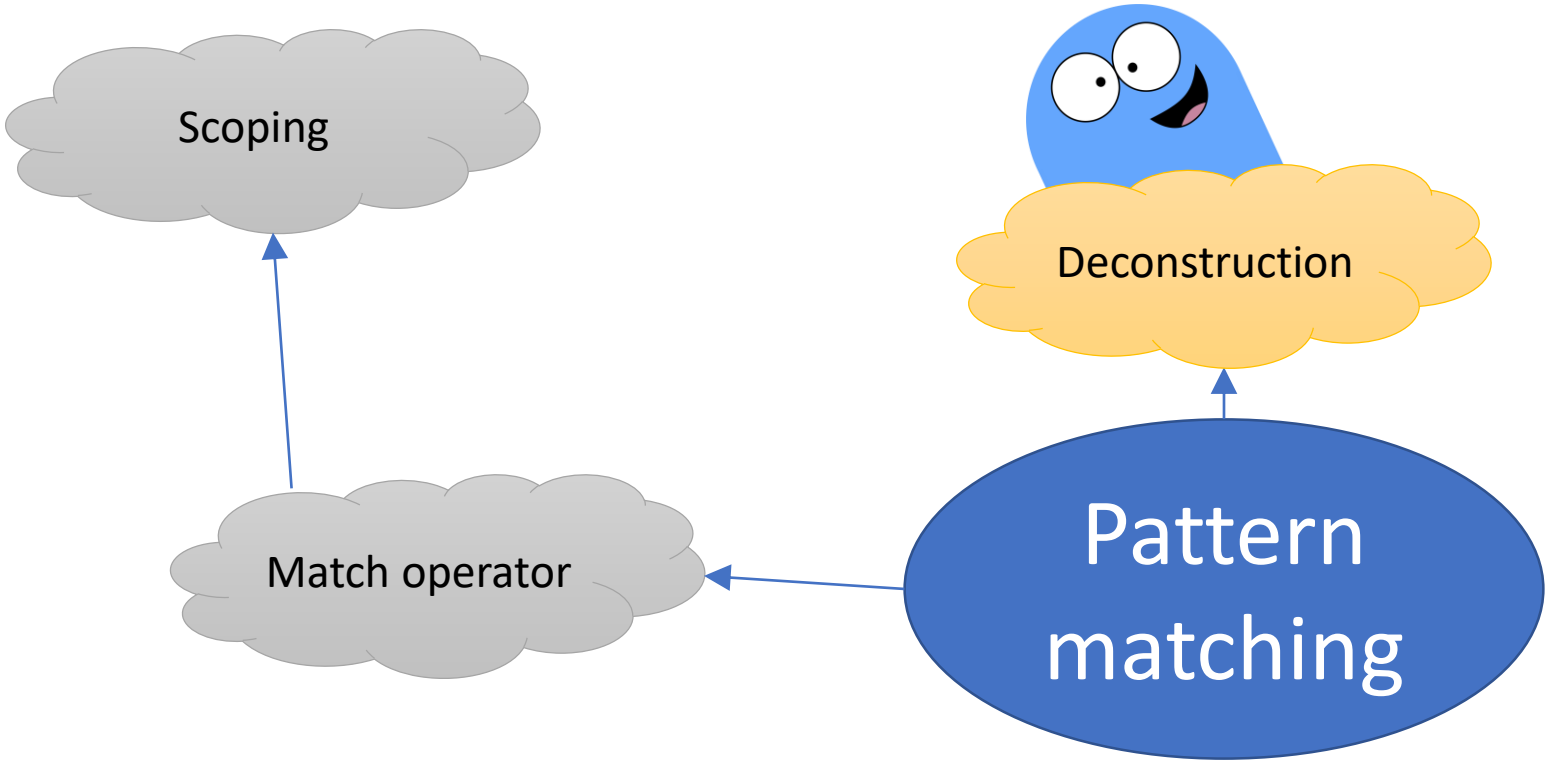
```
private Number n = 10;

void test(Object obj) {
  if(!(obj instanceof Number n) || n.longValue() == 0) {
    System.out.println(n.longValue());
  } else {
    System.out.println(n.longValue());
  }
}
```



```
void test(Object obj) {
  if(obj instanceof Number) {
    System.out.println(obj.longValue());
  }
}
```



Smart cast!


Scoping
Deconstruction
Match operator
Pattern matching

```
void printShape(Shape shape) {
  if (shape instanceof Line(Point from, Point to)) {
    System.out.println("Line: "+from+" - "+to);
  }



}
```

```
void printShape(Shape shape) {
  if (shape instanceof Line(Point from, Point to)) {
    System.out.println("Line: "+from+" - "+to);
  } else if (shape instanceof Circle(Point(int x, int y), int radius)) {
    System.out.println("Circle with center "+x+", "+y+" and radius "+radius);
  } else {
    System.out.println("Unknown shape");
  }
}
```

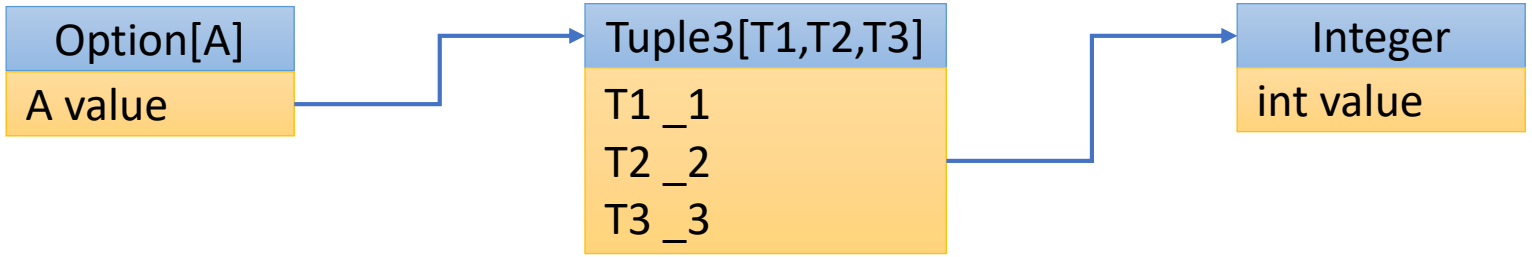
unapply

```
object Person {
  def apply(name: String, age: Int, address: Address) =
    new Person(name, age, address)
  def unapply(p: Person): Option[Tuple3[String,Int,Address]] =
    Some((p.name, p.age, p.address))
  ...
}
```

Programming Scala, 2nd Edition by Alex Payne, Dean Wampler
https://www.oreilly.com/library/view/programming-scala-2nd/9781491950135/ch04.html

unapply

```
object Person {
  def apply(name: String, age: Int, address: Address) =
    new Person(name, age, address)
  def unapply(p: Person): Option[Tuple3[String,Int,Address]] =
    Some((p.name, p.age, p.address))
  ...
}
```



Programming Scala, 2nd Edition by Alex Payne, Dean Wampler
https://www.oreilly.com/library/view/programming-scala-2nd/9781491950135/ch04.html

Deconstruct

```
public class User
{
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public int Age { get; set; }

  public void Deconstruct(out string firstName, out string lastName)
  {
    firstName = FirstName;
    lastName = LastName;
  }

  public void Deconstruct(out string firstName, out string lastName, out int age)
  {
    firstName = FirstName;
    lastName = LastName;
    age = Age;
  }
}
```

Deconstructors for non-tuple types in C# 7.0
https://andrewlock.net/deconstructors-for-non-tuple-types-in-c-7-0/

componentX

```
class Point(_x: Double, _y: Double) {
    var x: Double = _x
    var y: Double = _y

    operator fun component1() = x
    operator fun component2() = y
}
```

componentX

```
class Point(_x: Double, _y: Double) {
    var x: Double = _x
    var y: Double = _y

    fun rotate(angle: Double) {
        val _x = x*cos(angle)-y*sin(angle)
        val _y = x*sin(angle)+y*cos(angle)
        x = _x;
        y = _y;
    }

    operator fun component1() = x
    operator fun component2() = y
}
```

componentX

```
class Point(_x: Double, _y: Double) {
    var x: Double = _x
    var y: Double = _y

    fun rotate(angle: Double) {
        synchronized(this) {
            val _x = x * cos(angle) - y * sin(angle)
            val _y = x * sin(angle) + y * cos(angle)
            x = _x;
            y = _y;
        }
    }

    operator fun component1() = synchronized(this) { x }
    operator fun component2() = synchronized(this) { y }
}
```

Язык	Частичное применение	JVM?	Вложенность объектов	Атомарность	Перегрузка деконструкторов	Имена в объявлении
Scala	✓	✓	1-3	✓	✗	✗
C#	✗	✗	0	✓	+/-	✓
Kotlin	✗	✓	0	✗	✗	✗
Хочется	✓	✓	0	✓	✓	✓

```
void construct(String path, URI uri) {
  File f1 = new File(path);
  File f2 = new File(uri);
}
```

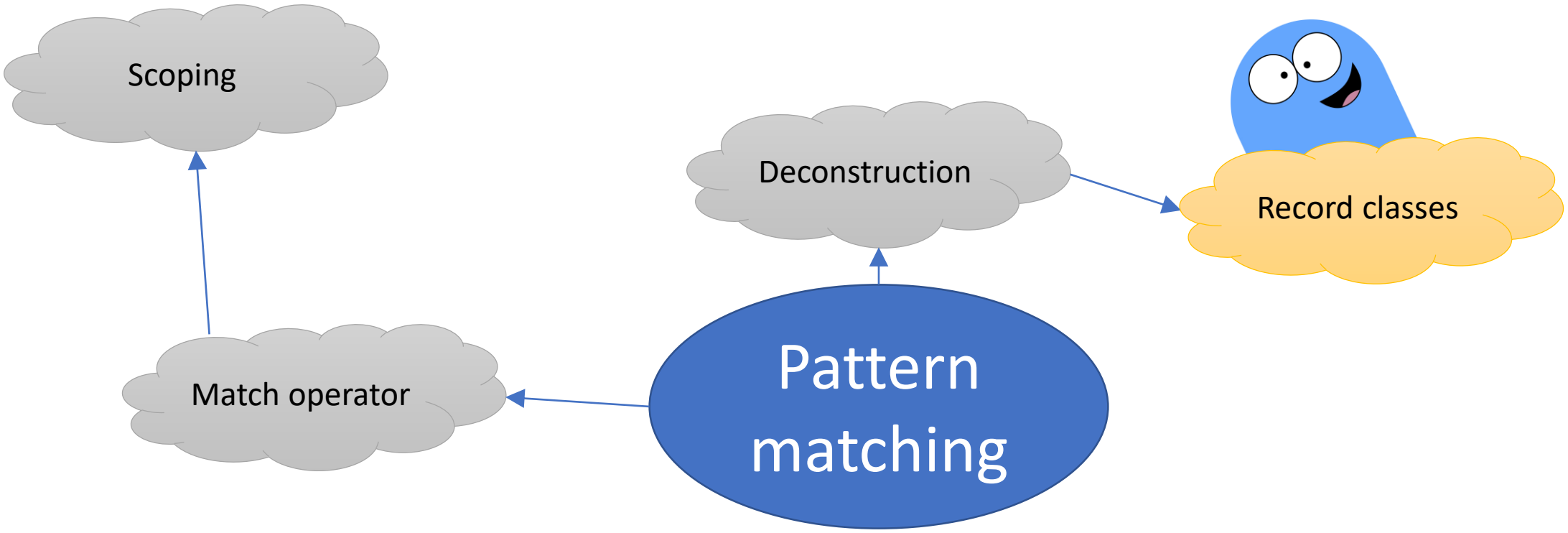
```
void construct(String path, URI uri) {
  File f1 = new File(path);
  File f2 = new File(uri);
}

void deconstruct(File f1, File f2) {
  if(f1 instanceof File(String path)) System.out.println(path);
  if(f2 instanceof File(URI uri)) System.out.println(uri);
}
```

http://cr.openjdk.java.net/~briangoetz/amber/pattern-match-translation.html

```
public interface __Pattern<T> {
  /* A method handle used to preprocess a target into an intermediate carrier. The method handle
     accepts a match target and returns the intermediate carrier. If the isCarrierFree() method
     returns true, then this method need not be called, and null can be used for the carrier
     in other method handle */
  MethodHandle preprocess();
  /* A method handle used to determine if the match succeeds. It accepts the match target and
     the intermediate carrier returned by preprocess(), and returns a boolean indicating whether
     the match was successful. If the pattern is declared to always match, then this method
     need not be called. */
  MethodHandle predicate();
  /* A method handle to return the i'th component of a successful match. It accepts the match
     target and the intermediate carrier returned by preprocess(), and returns the component. */
  MethodHandle component(int i);
  /* Indicates that this pattern does not make use of an intermediate carrier, and that the
     preprocess() method handle is a no-op. Combinators exploit carrier freedom to reduce
     unnecessary allocation. */
  boolean isCarrierFree();
  /* Returns the pattern descriptor, which is a MethodType whose return type is the match target,
     and whose parameter types are the components of the match. */
  MethodType descriptor();
}
```

```
class Point {
  private int x, y;

  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  extractor Point(int x, int y) {
    x = this.x;
    y = this.y;
  }
}
```

```
class Point {
  private int x, y;

  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  extractor Point(int x, int y) {
    x = this.x;
    y = this.y;
  }
}
```

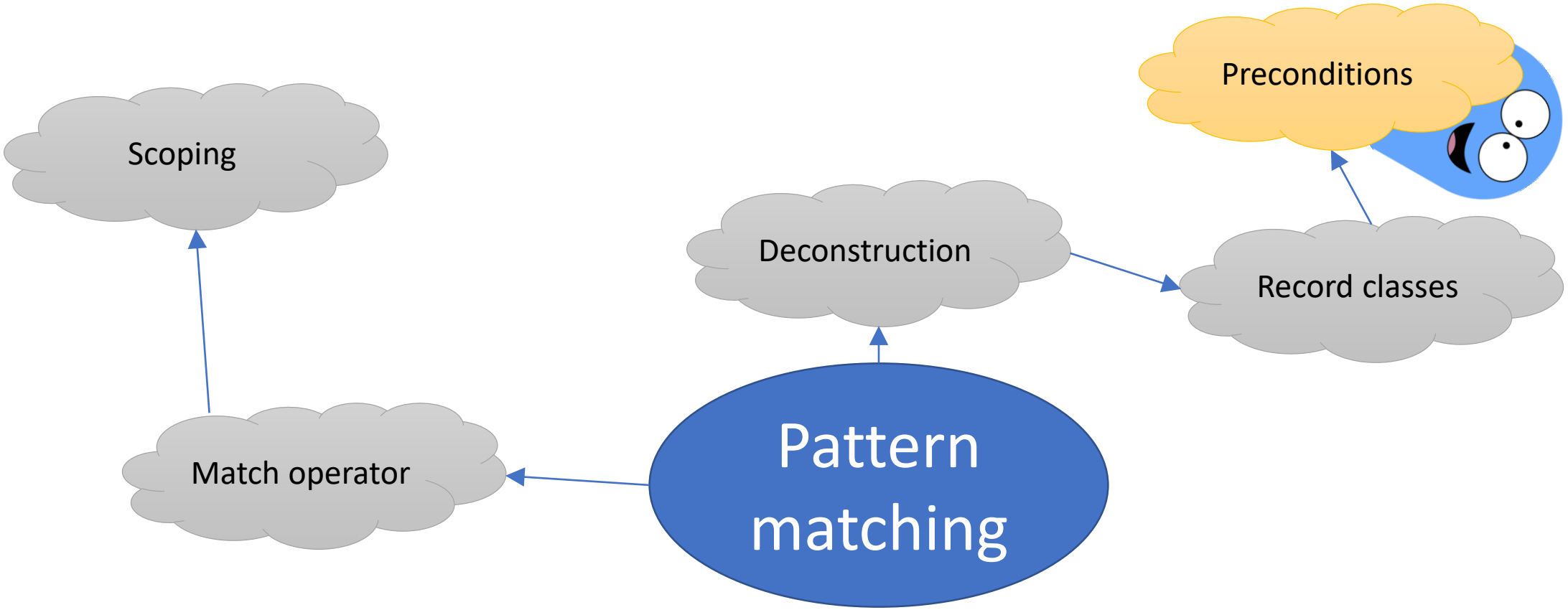

Scoping
Deconstruction
Record classes
Match operator
Pattern matching

```
record Point(int x, int y);
```

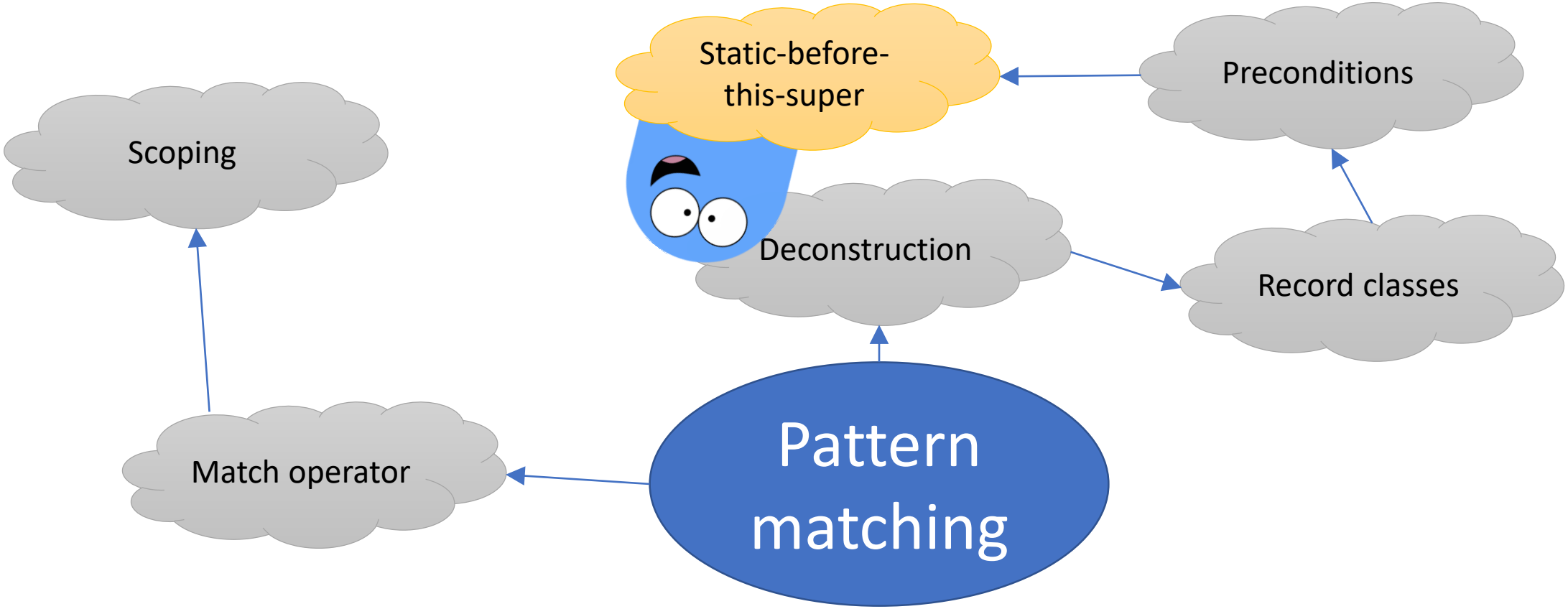
- ✓ Как назвать геттеры? getX()? Или просто x()?
- ✓ Нужна ли изменяемость?
  - Генерировать ли сеттеры (и как их назвать)?
  - Что будет по умолчанию? Писать final для неизменяемых или non-final для изменяемых?
  - Генерировать ли конструктор по умолчанию?
  - Генерировать ли clone()?
- ✓ Можно ли добавлять дополнительные поля (кэш для вычисляемого значения)?
- ✓ Как установить инварианты (requireNonNull, a < b и т. д.)?
- ✓ Как нормализовать? (a < 0 -> a = 0)
- ✓ В каком порядке генерировать поля в equals?
- ✓ Как конкретно сравнивать?
  - Поле-массив: Arrays.equals?
  - Поле-double: ==? Double.compare?
- ✓ Может ли record реализовать интерфейс?
- ✓ Может ли record расширить (абстрактный) класс с полями?
- ✓ Можно ли расширить record другим record?
- ✓ Можно ли расширить record обычным классом?

- ✓ Как назвать геттеры? getX()? Или просто x()?
- ✓ Нужна ли изменяемость?
  - Генерировать ли сеттеры (и как их назвать)?
  - Что будет по умолчанию? Писать final для неизменяемых или non-final для изменяемых?
  - Генерировать ли конструктор по умолчанию?
  - Генерировать ли clone()?
- ✓ Можно ли добавлять дополнительные поля (кэш для вычисляемого значения)?
- ✓ Как установить инварианты (requireNonNull, a < b и т. д.)?
- ✓ Как нормализовать? (a < 0 -> a = 0)
- ✓ В каком порядке генерировать поля в equals?
- ✓ Как конкретно сравнивать?
  - Поле-массив: Arrays.equals?
  - Поле-double: ==? Double.compare?
- ✓ Может ли record реализовать интерфейс?
- ✓ Может ли record расширить (абстрактный) класс с полями?
- ✓ Можно ли расширить record другим record?
- ✓ Можно ли расширить record обычным классом?

- ✓ Как назвать геттеры? getX()? Или просто x()?
- ✓ Нужна ли изменяемость?
  - Генерировать ли сеттеры (и как их назвать)?
  - Что будет по умолчанию? Писать final для неизменяемых или non-final для изменяемых?
  - Генерировать ли конструктор по умолчанию?
  - Генерировать ли clone()?
- ✓ Можно ли добавлять дополнительные поля (кэш для вычисляемого значения)?
- ✓ Как установить инварианты (requireNonNull, a < b и т. д.)?
- ✓ Как нормализовать? (a < 0 -> a = 0)
- ✓ В каком порядке генерировать поля в equals?
- ✓ Как конкретно сравнивать?
  - Поле-массив: Arrays.equals?
  - Поле-double: ==? Double.compare?
- ✓ Может ли record реализовать интерфейс?
- ✓ Может ли record расширить (абстрактный) класс с полями?
- ✓ Можно ли расширить record другим record?
- ✓ Можно ли расширить record обычным классом?


Preconditions
Scoping
Deconstruction
Record classes
Match operator
Pattern matching

```
record Range(int lo, int hi) {

  // Explicit default constructor
  @Override
  public Range(int lo, int hi) {
    // validation logic
    if (lo > hi)
      throw new IllegalArgumentException(...);

    // delegate to default constructor
    default.this(lo, hi);
  }
}
```


Static-before-this-super
Preconditions
Scoping
Deconstruction
Record classes
Match operator
Pattern matching

```
class Point {
  int x, y;

  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}

class NonNullPoint extends Point {
  NonNullPoint(int x, int y) {
    if (x == 0 && y == 0) throw new IllegalArgumentException();
    super(x, y);
  }
}
```

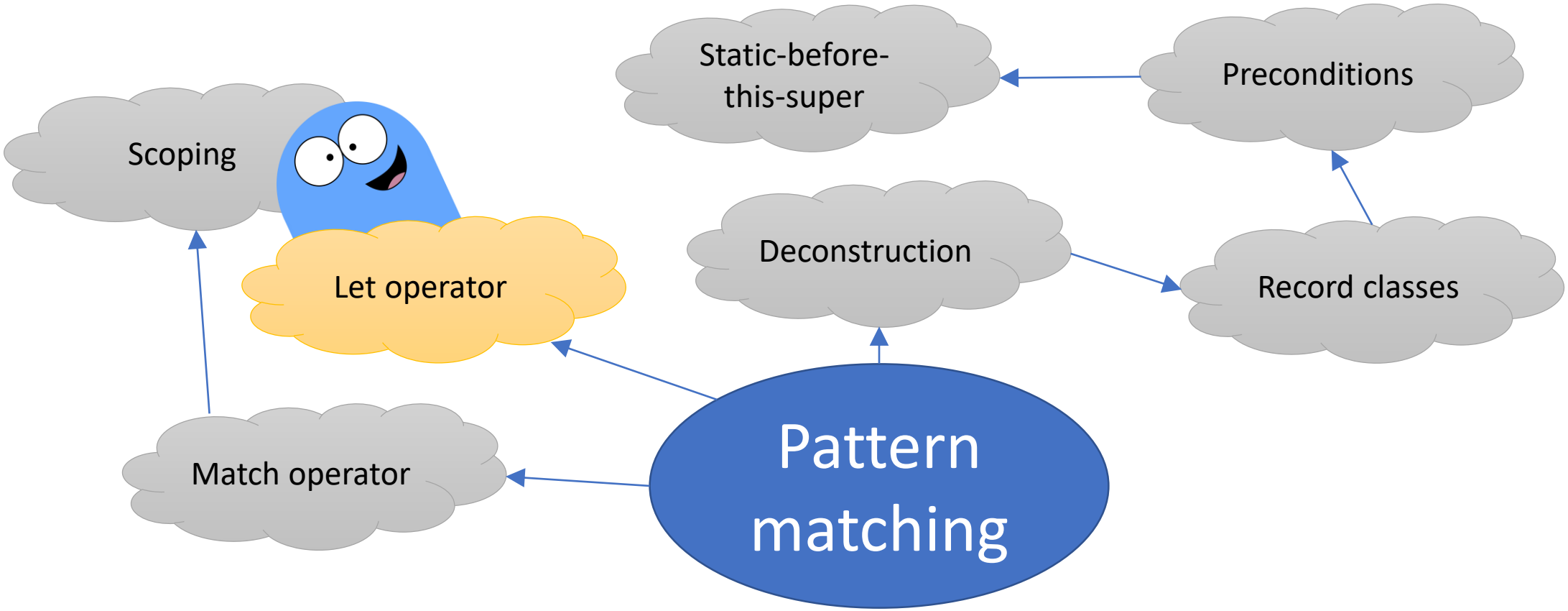
http://hg.openjdk.java.net/amber/amber/branches -> stats-before-this-super

```
record Range(int lo, int hi) {

  // Explicit default constructor
  @Override
  public Range(int lo, int hi) {
    // validation logic
    if (lo > hi)
      throw new IllegalArgumentException(...);

    // delegate to default constructor
    default.this(lo, hi);
  }
}
```

```
record Range(int lo, int hi)
  requires (lo <= hi) {
}
```

```
record Range(int lo, int hi) {

  // Explicit default constructor
  @Override
  public Range(int lo, int hi) {
    // validation logic
    if (lo > hi) {
      int tmp = lo;
      lo = hi;
      hi = tmp;
    }

    // delegate to default constructor
    default.this(lo, hi);
  }
}
```

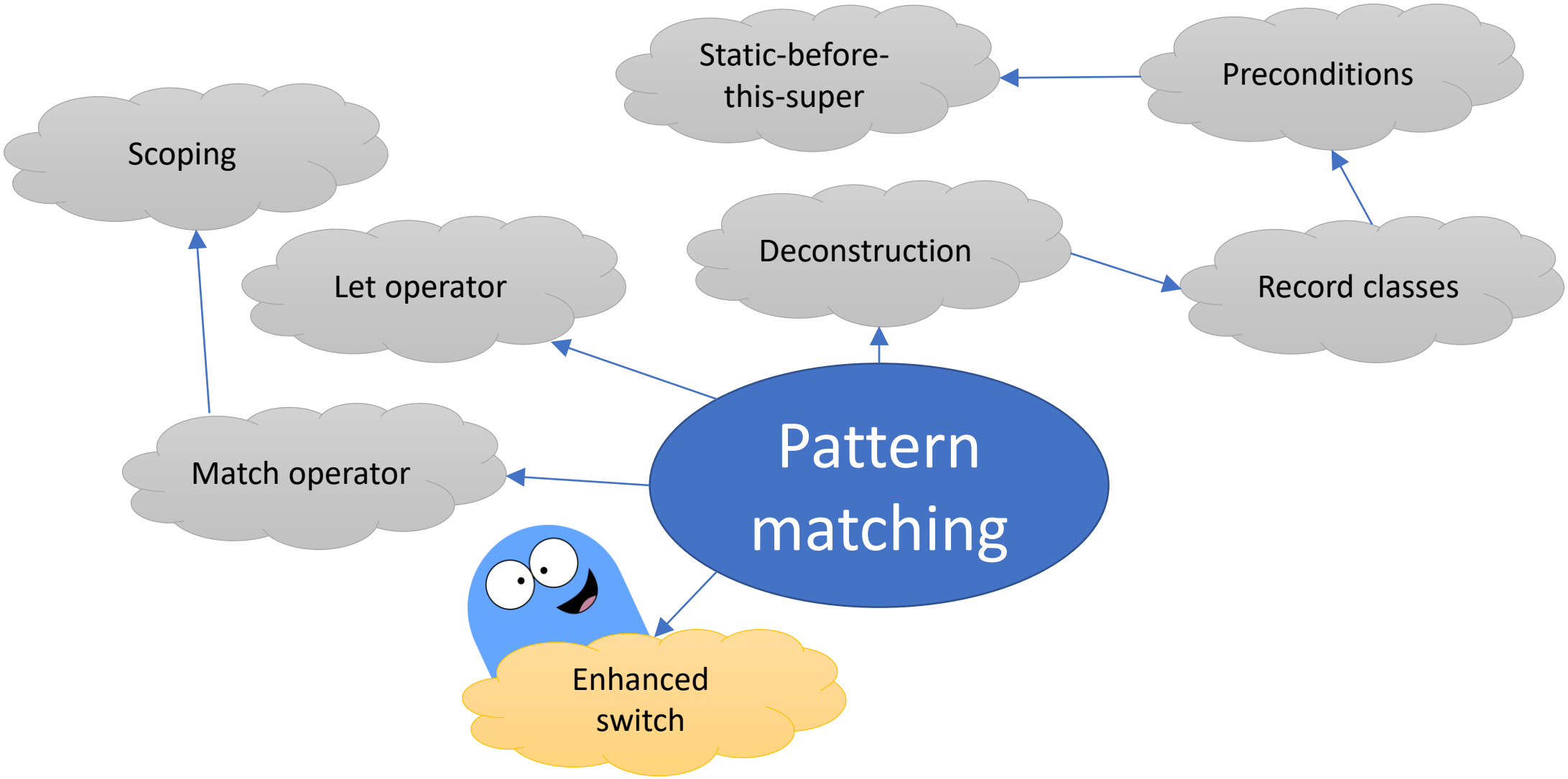
```
void test(Point p) {
  if(p instanceof Point(int x, int y)) {
    System.out.println(x+"; "+y);
  }
}
```


Static-before-this-super
Preconditions
Scoping
Let operator
Deconstruction
Record classes
Match operator
Pattern matching

```
void test(Point p) {
  let Point(int x, int y) = p;
  System.out.println(x+"; "+y);
}
```

```
void test(Object p) {
  let Point(int x, int y) = p;
  System.out.println(x+"; "+y);
}
```

```
void test(Object p) {
  let Point(int x, int y) = p
  else throw new IllegalArgumentException("Oops");
  System.out.println(x+"; "+y);
}
```

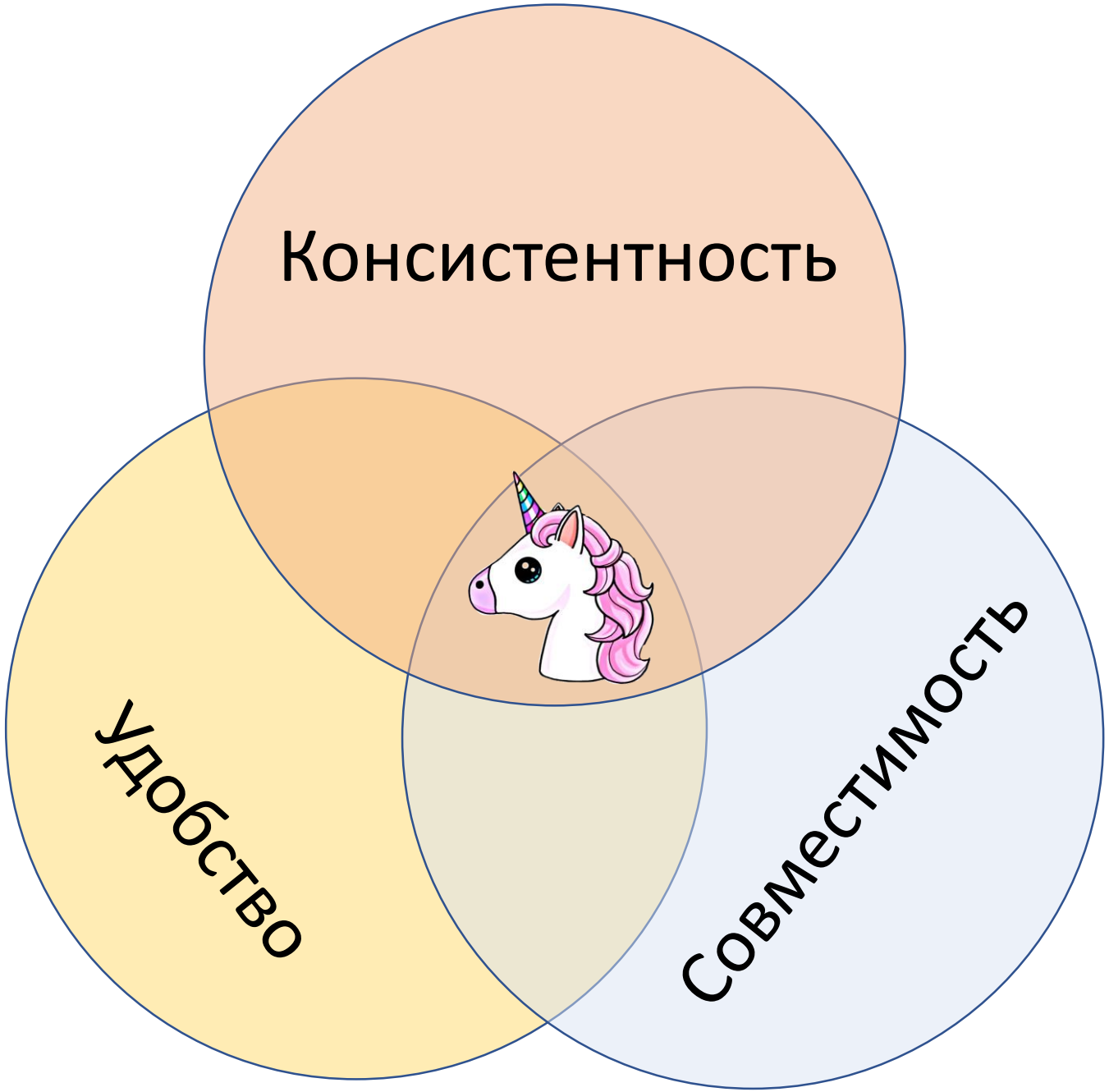

Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Enhanced switch

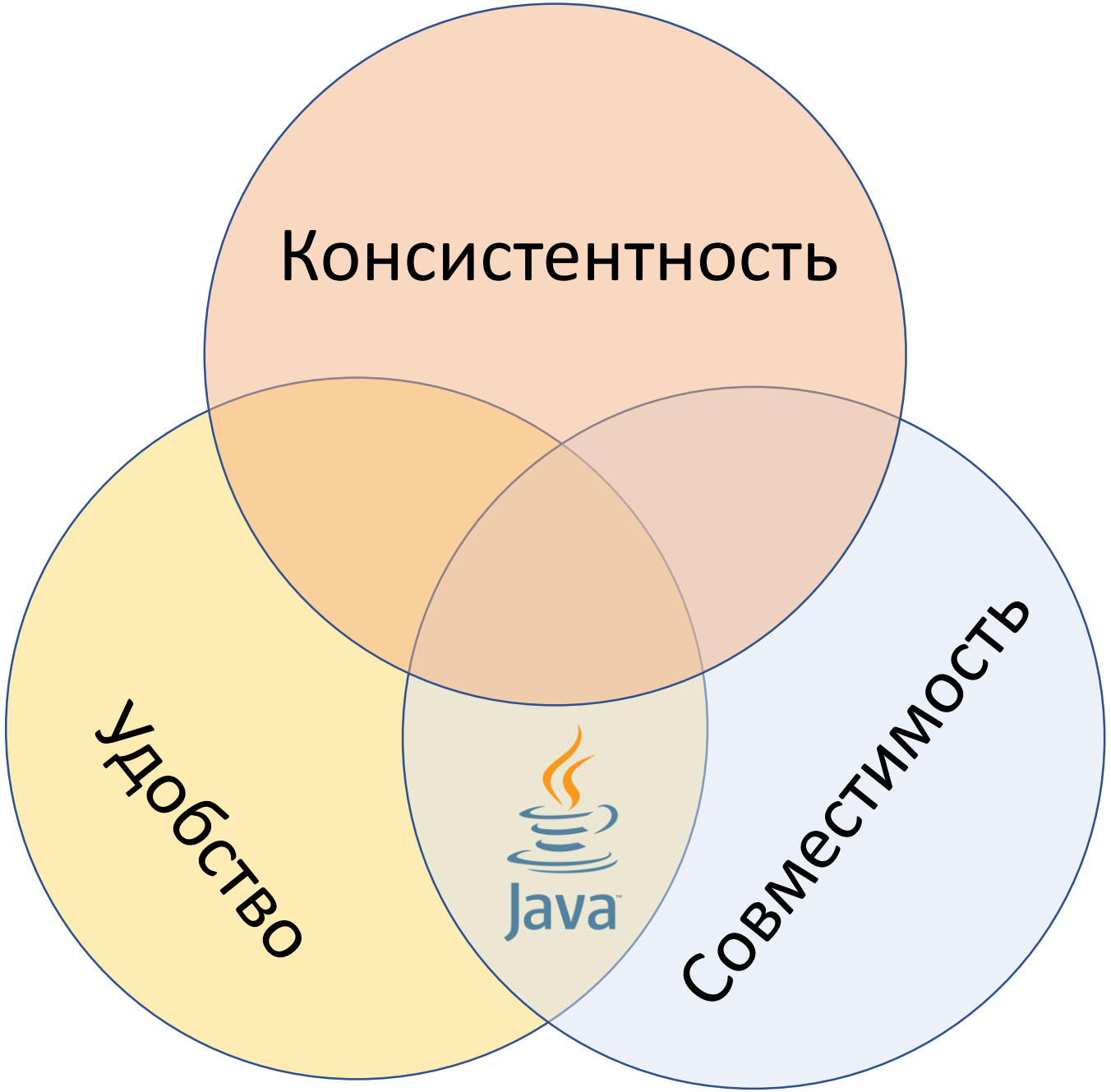
```
void printObject(Object obj) {
  switch(obj) {
    case String s:
      System.out.println("Строка-шмока: "+s.trim());
      break;
    case Integer i:
      System.out.println("Целое-шмелое: "+i);
      break;
    case Number n:
      System.out.println("Число-шмисло: "+n);
      break;
    default:
      System.out.println("Что-то с чем-то");
  }
}
```

```
void printObject(Object obj) {
  switch(obj) {
    case String s:
      System.out.println("Строка-шмока: "+s.trim());
      break;
    case Number n:
      System.out.println("Число-шмисло: "+n);
      break;
    case Integer i:
      System.out.println("Целое-шмелое: "+i);
      break;
    default:
      System.out.println("Что-то с чем-то");
  }
}
```

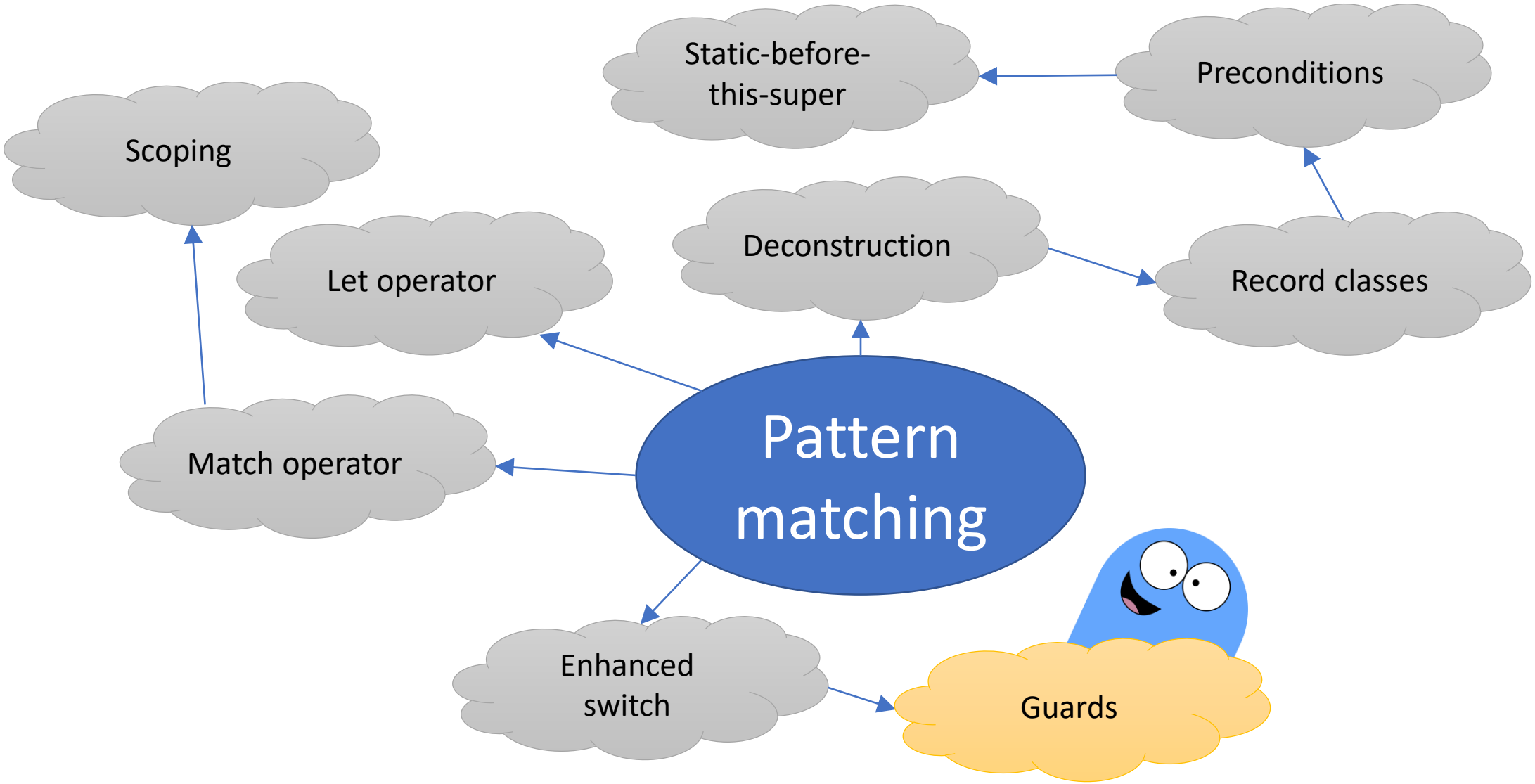
```
void printObject(Object obj) {
  switch(obj) {
    case String s:
      System.out.println("Строка-шмока: "+s.trim());
      break;
    case Number n:
      System.out.println("Число-шмисло: "+n);
      break;
    case Integer i:
      System.out.println("Целое-шмелое: "+i);
      break;
    default:// == case _:
      System.out.println("Что-то с чем-то");
  }
}
```

```
void switchTest(int i) {
  switch (i) {
    case 1:
      System.out.println("One");
      break;
    default:
      System.out.println("Other");
      break;
    case 2:
      System.out.println("Two");
      break;
  }
}
```


Консистентность
Удобство
Совместимость


Консистентность
удобство
Совместимость
Java


Не мы такие, жизнь такая.

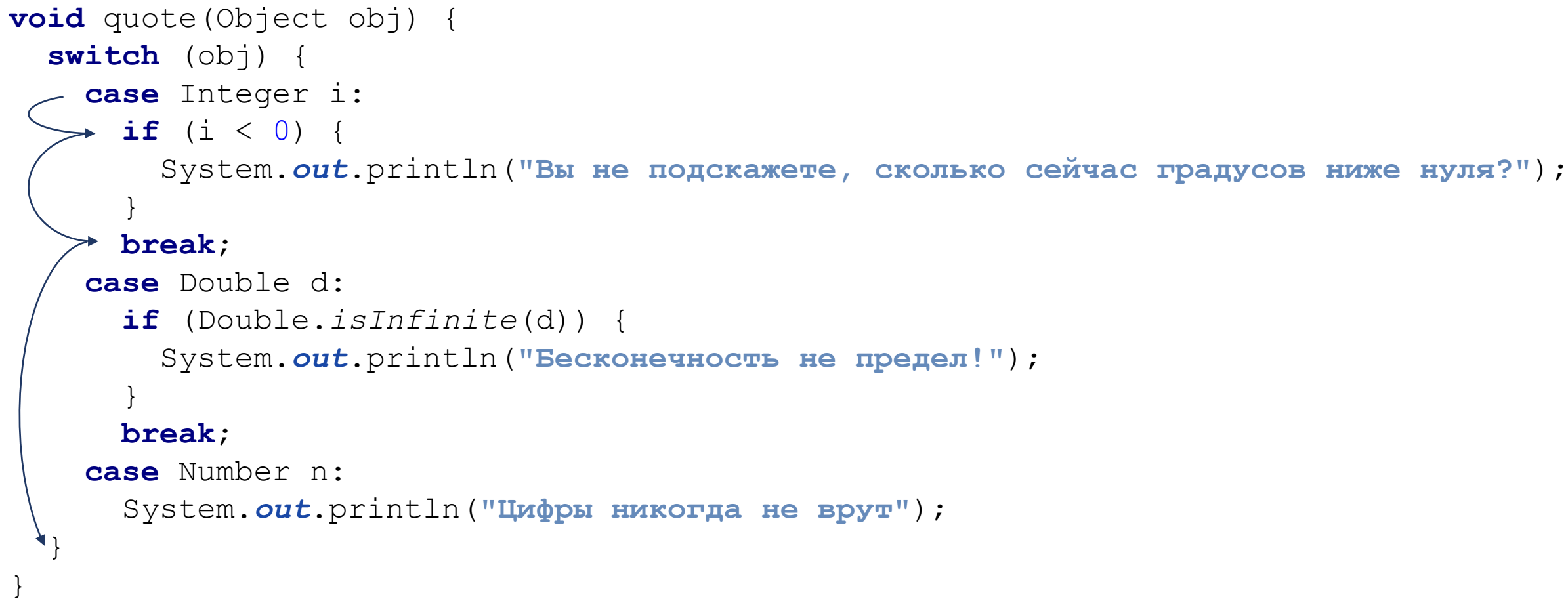

Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Enhanced switch
Guards

```
void quote(Object obj) {
  switch (obj) {
    case Integer i where i < 0:
      System.out.println("Вы не подскажете, сколько сейчас градусов ниже нуля?");
      break;
    case Double d where Double.isInfinite(d):
      System.out.println("Бесконечность не предел!");
      break;
    case Number n:
      System.out.println("Цифры никогда не врут");
  }
}
```

obj = 5

```
void quote(Object obj) {
  switch (obj) {
    case Integer i where i < 0:
      System.out.println("Вы не подскажете, сколько сейчас градусов ниже нуля?");
      break;
    case Double d where Double.isInfinite(d):
      System.out.println("Бесконечность не предел!");
      break;
    case Number n:
      System.out.println("Цифры никогда не врут");
  }
}
```

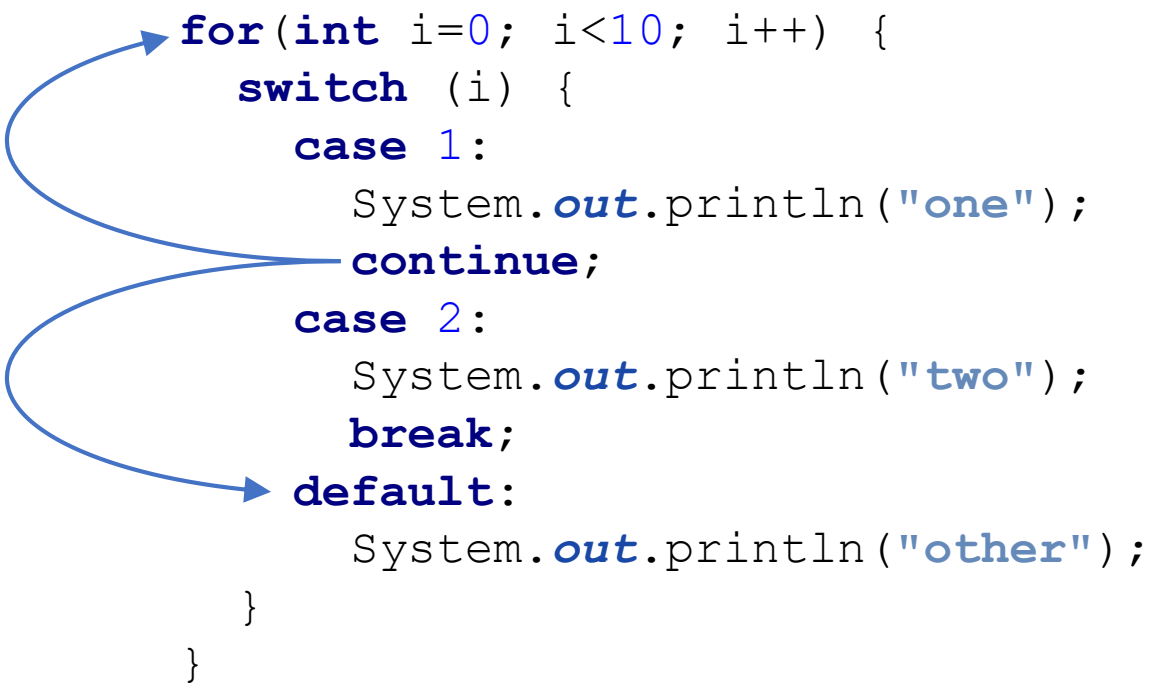
obj = 5

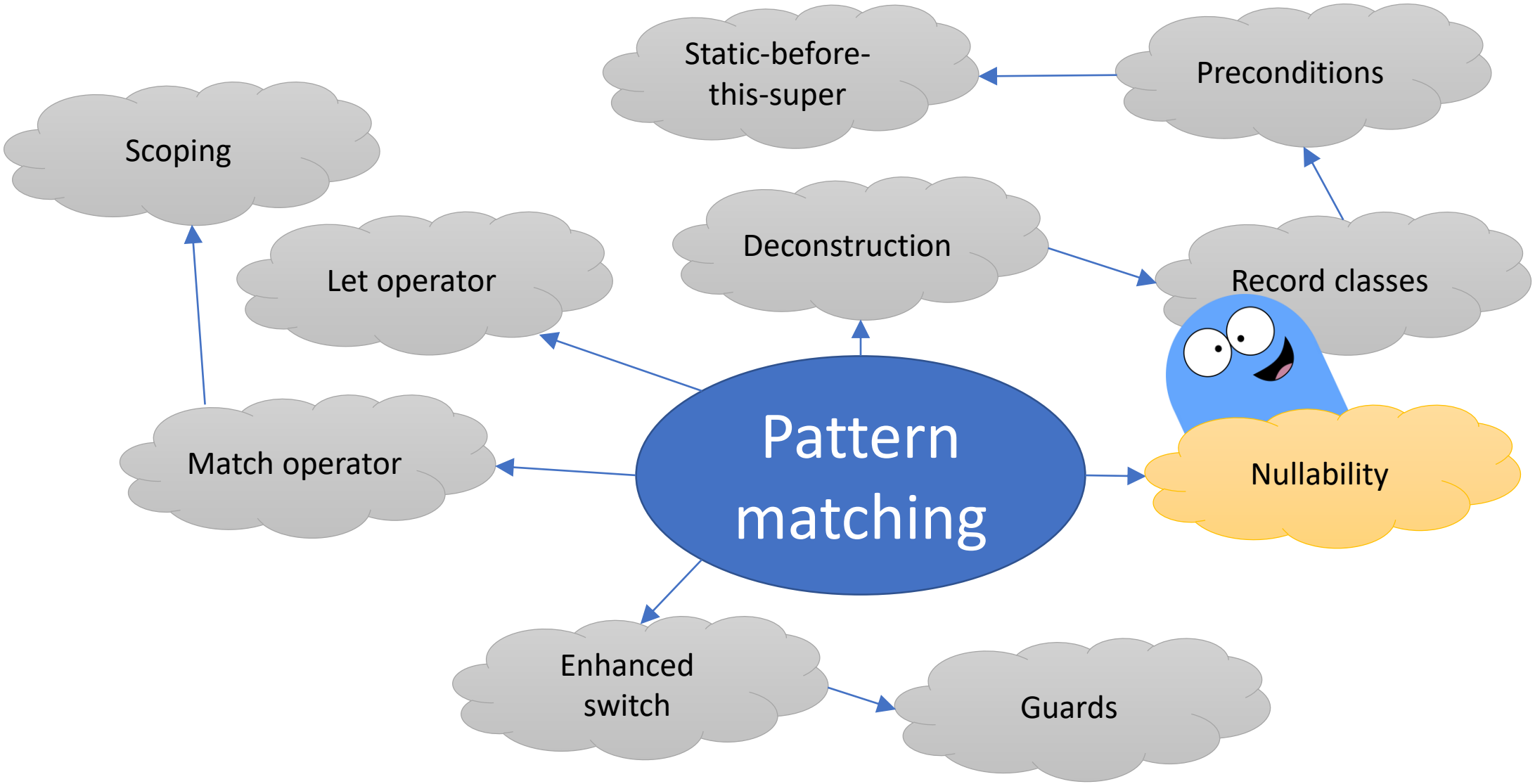


```
void quote(Object obj) {
  switch (obj) {
    case Integer i:
      if (i < 0) {
        System.out.println("Вы не подскажете, сколько сейчас градусов ниже нуля?");
      }
      break;
    case Double d:
      if (Double.isInfinite(d)) {
        System.out.println("Бесконечность не предел!");
      }
      break;
    case Number n:
      System.out.println("Цифры никогда не врут");
  }
}
```

`Continue` to the rescue!

```
void quote(Object obj) {
  switch (obj) {
    case Integer i:
      if (i >= 0) continue;
      System.out.println("Вы не подскажете, сколько сейчас градусов ниже нуля?");
      break;
    case Double d:
      if (Double.isFinite(d)) continue;
      System.out.println("Бесконечность не предел!");
      break;
    case Number n:
      System.out.println("Цифры никогда не врут");
  }
}
```


for(int i=0; i<10; i++) {
switch (i) {
case 1:
System.out.println("one");
continue;
case 2:
System.out.println("two");
break;
default:
System.out.println("other");
}
}



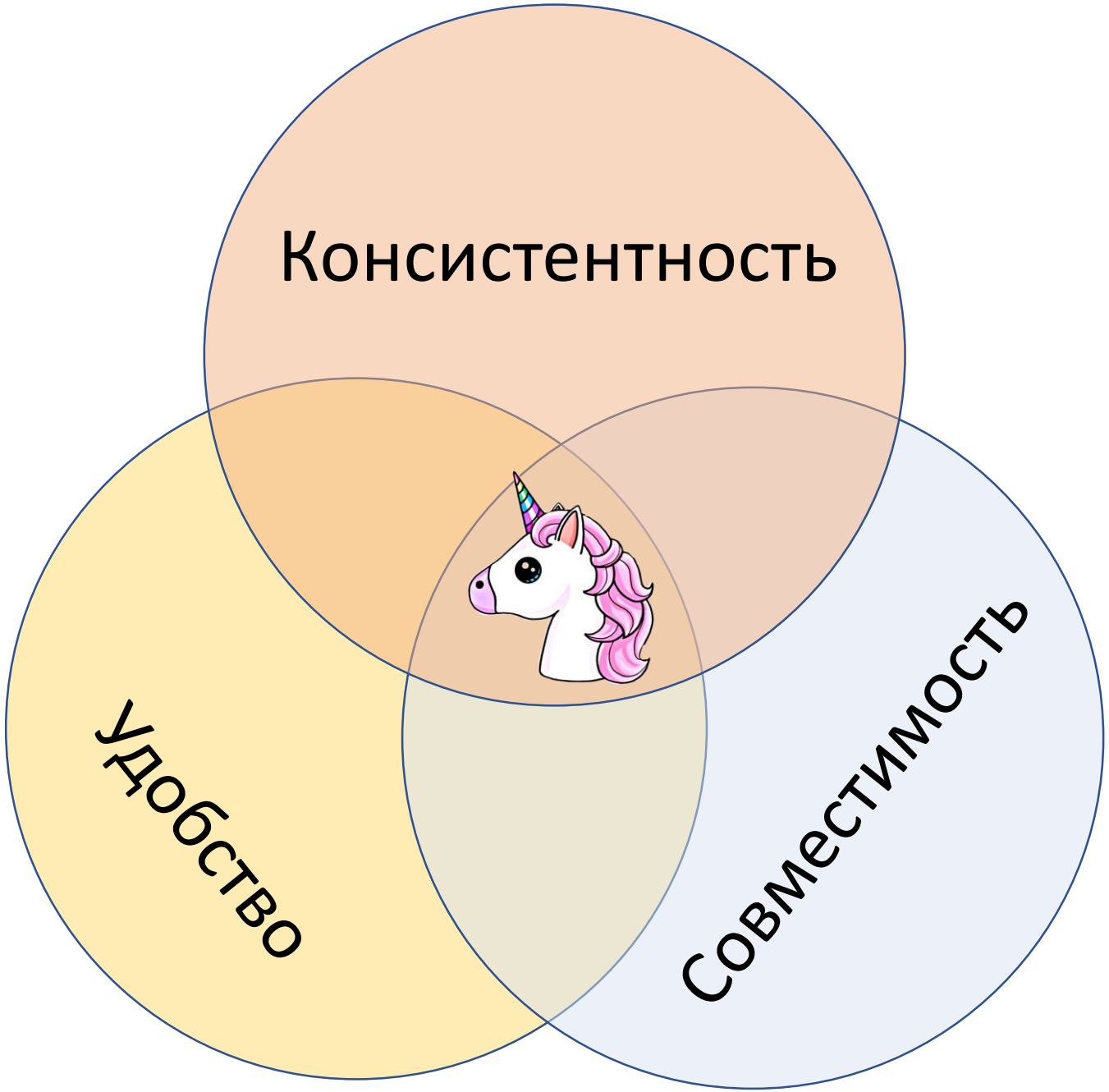
Матчит ли String s значение null?

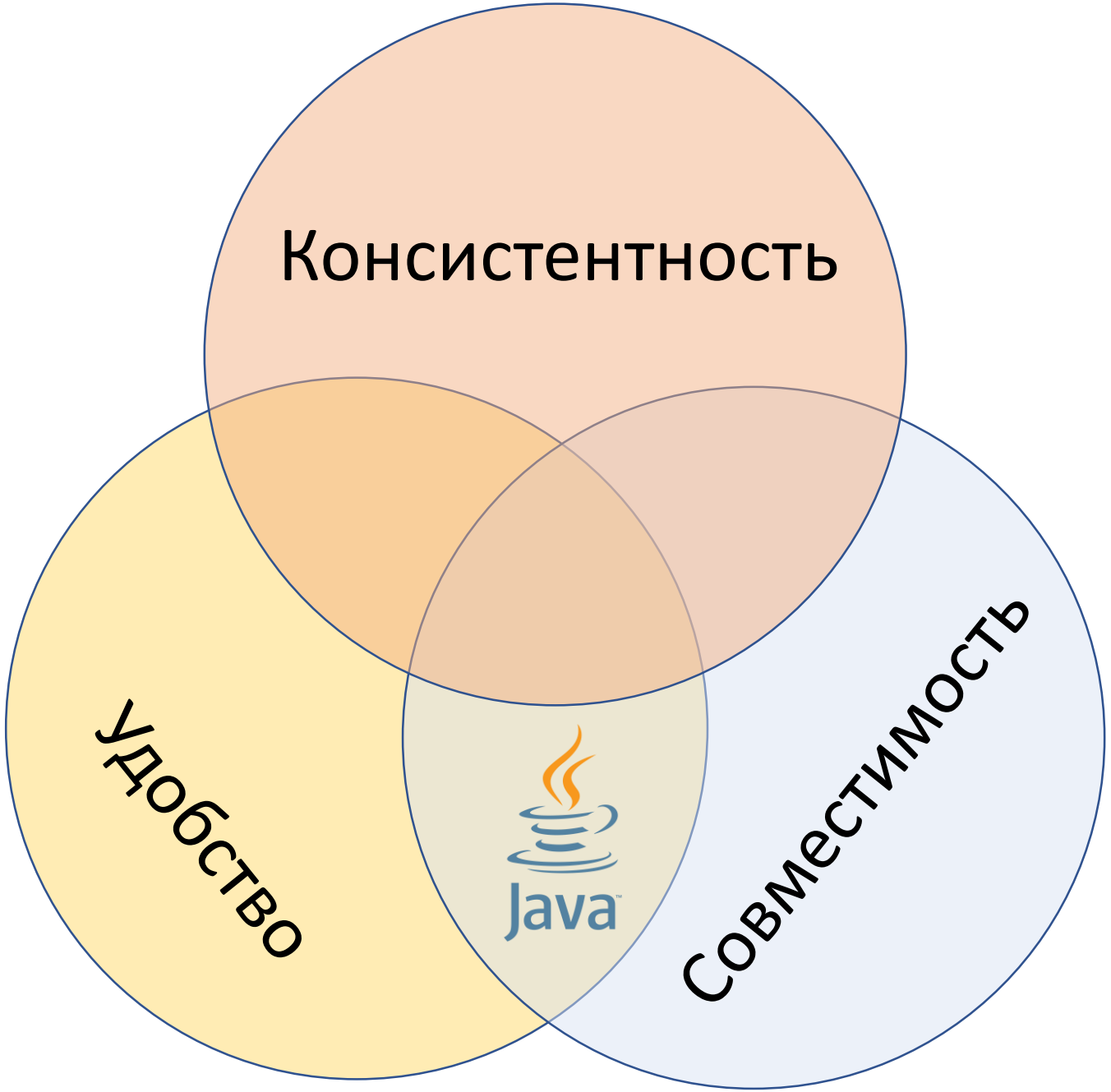
```java
if (obj instanceof String s) {
  System.out.println(s.length());
}
```

```java
if (obj instanceof String s) {
  System.out.println(s.length());
}

switch (obj) {
  case String s:  System.out.println("String "+s);  break;
  case Integer i: System.out.println("Integer "+i); break;
  default:        System.out.println("other");       break;
}
```

```
switch (str) {
  case "JPoint":
    System.out.println("Бордюр"); break;
  case "Joker":
    System.out.println("Поребрик"); break;
  default:// == case _:
    System.out.println("Загогулина вдоль дороги"); break;
}
```

```
switch (str) {
  case "JPoint":
    System.out.println("Бордюр"); break;
  case "Joker":
    System.out.println("Поребрик"); break;
  case null:
    System.out.println("Тлен"); break;
  default:// == case _:
    System.out.println("Загогулина вдоль дороги"); break;
}
```


Консистентность
Удобство
Совместимость


Консистентность
удобство
Совместимость
Java

```
switch (str) {
  case "JPoint":
    System.out.println("Бордюр"); break;
  case "Joker":
    System.out.println("Поребрик"); break;
  case null:
  default:// == case _:
    System.out.println("Загогулина вдоль дороги"); break;
}
```

```
switch (str) {
  case "JPoint":
    System.out.println("Бордюр"); break;
  case "Joker":
    System.out.println("Поребрик"); break;
  case _:
    System.out.println("Загогулина вдоль дороги"); break;
}
```

default – это...

default – это

    case Object _

default – это

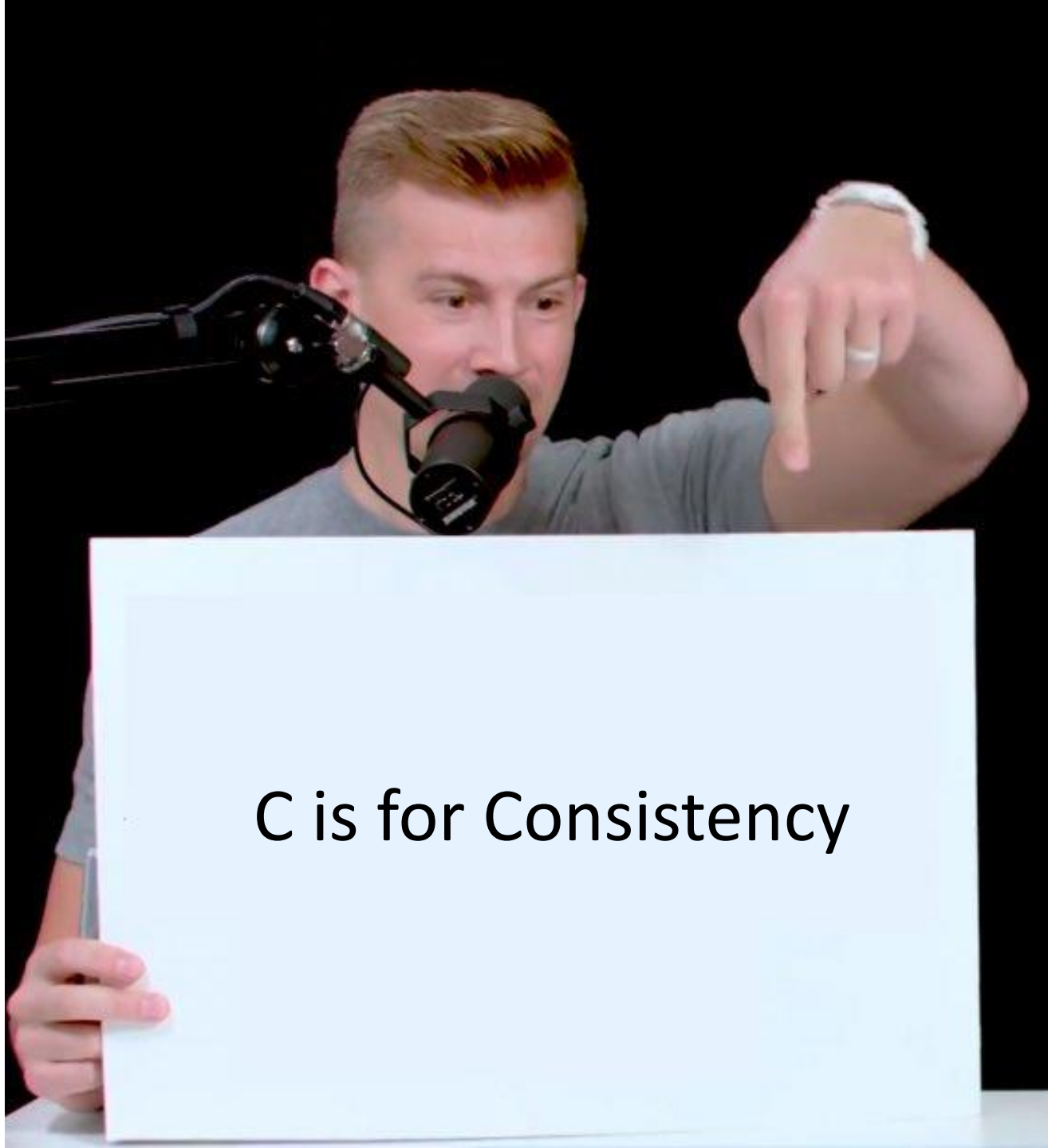
case Object _, если switch

не по примитиву;

case _, если switch

по примитиву.

default – это
    case Object _, если switch
        не по примитиву;
    case _, если switch
        по примитиву.



```
record Node(String name, Node left, Node right)
  requires name != null {
}
```

```
record Node(String name, Node left, Node right)
  requires name != null {

  static Node leaf(String name) {
    return new Node(name, null, null);
  }

}
```

```
record Node(String name, Node left, Node right)
  requires name != null {

  static Node leaf(String name) {
    return new Node(name, null, null);
  }

}

void printIfNode(Object obj) {
  if (obj instanceof Node(String name, Node left, Node right)){
    System.out.println("Name  = " + name);
    System.out.println("Left  = " + left);
    System.out.println("Right = " + right);
  }
}
```


Реабилитация-Магнитогорск.рф
НАРКОМАНИЯ
ВЫХОД ЕСТЬ
АЛКОГОЛИЗМ
Бесплатные консультации родителям
Телефон доверия
43-12-91
8-922-742-55-33

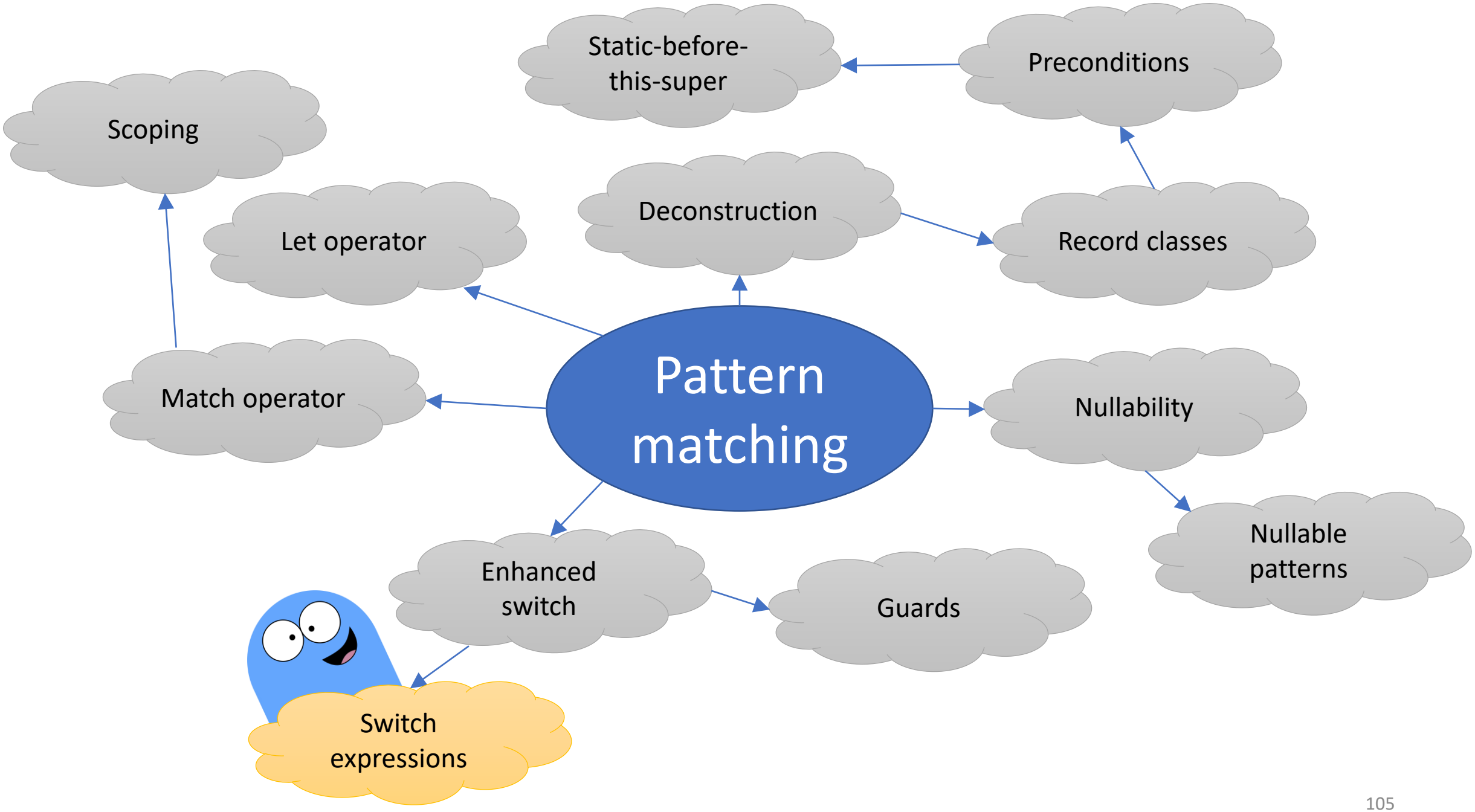
НУЛЛОМАНИЯ
ВЫХОД ЕСТЬ
NULLABLE PATTERNS

Бесплатные консультации тимлидам

Телефон доверия 03

```
void printIfNode(Object obj) {
  if (obj instanceof Node(String name, Node? left, Node? right)){
    System.out.println("Name  = " + name);
    System.out.println("Left  = " + left);
    System.out.println("Right = " + right);
  }
}
```

```
void printIfNode(Object obj) {
  if (obj instanceof Node(String name, Node? left, Node? right)){
    System.out.println("Name  = " + name);
    System.out.println("Left  = " + left);
    System.out.println("Right = " + right);
  }
}
```


Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Nullability
Enhanced switch
Guards
Nullable patterns
Switch expressions

```
void printObject(Object obj) {
  switch(obj) {
    case String s:
      System.out.println("Строка-шмока: "+s.trim());
      break;
    case Integer i:
      System.out.println("Целое-шмелое: "+i);
      break;
    case Number n:
      System.out.println("Число-шмисло: "+n);
      break;
    default:
      System.out.println("Что-то с чем-то");
  }
}
```

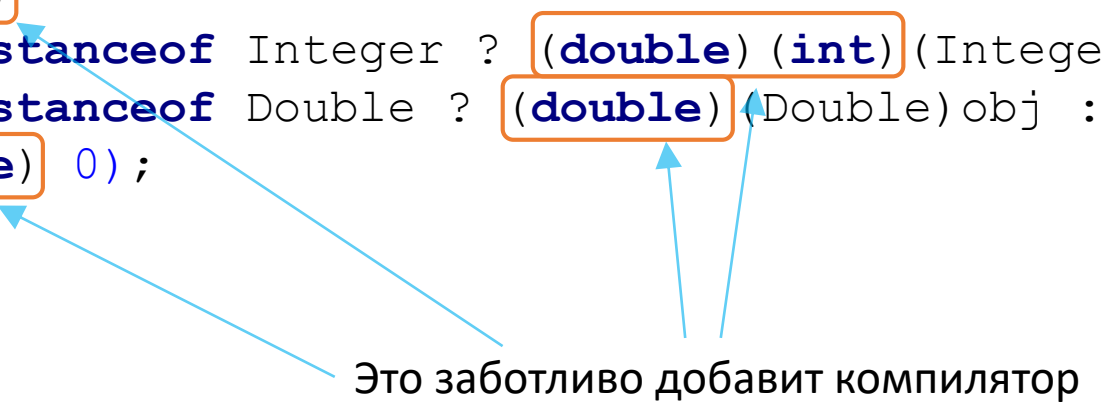
```
void printObject(Object obj) {
  System.out.println(switch (obj) {
    case String s  -> "Строка-шмока: " + s.trim();
    case Integer i -> "Целое-шмелое: " + i;
    case Number n  -> "Число-шмисло: " + n;
    default        -> "Что-то с чем-то";
  });
}
```

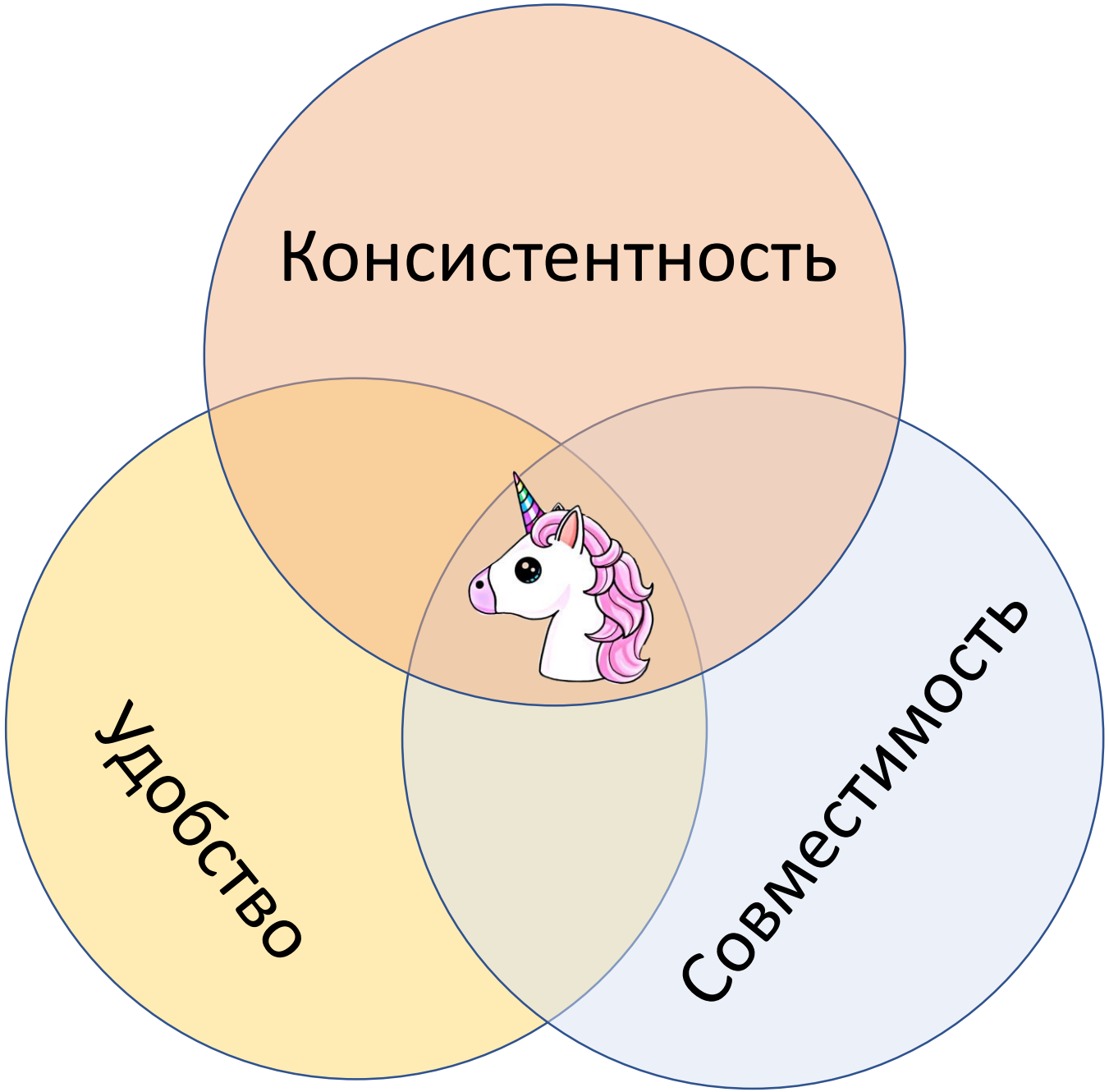
```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> 0;
  };
}
```

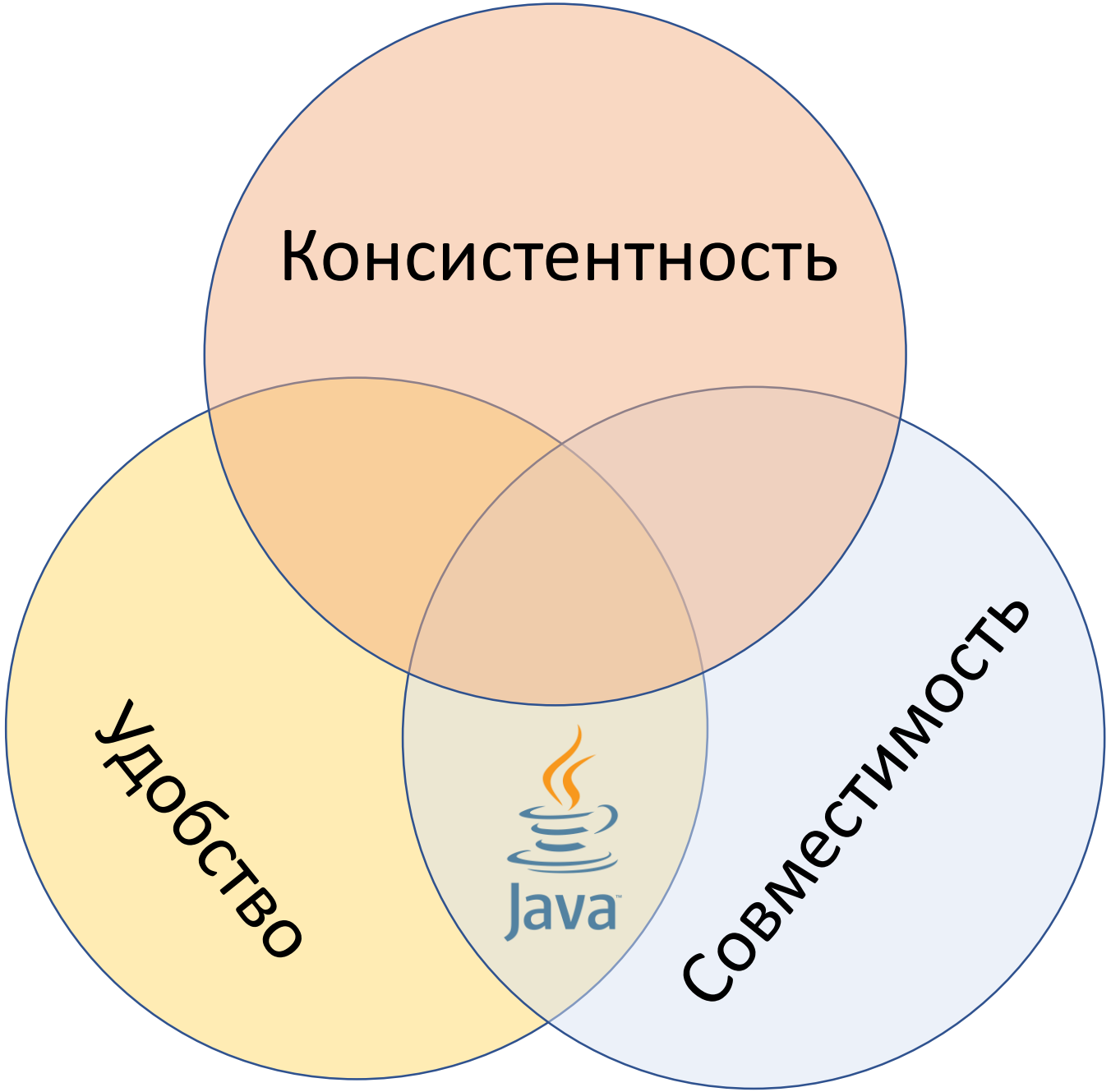
```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> 0;
  };
}

void test(Object obj) {
  Number n = obj instanceof Integer ? (Integer)obj :
             obj instanceof Double ? (Double)obj :
             0;
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> 0;
  };
}
```

```
void test(Object obj) {
  Number n = (Double)
             (obj instanceof Integer ? (double)(int)(Integer)obj :
              obj instanceof Double ? (double)(Double)obj :
              (double) 0);
}
```




Консистентность
Удобство
Совместимость


Консистентность
удобство
Совместимость
Java

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> …
  };
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> throw new IllegalArgumentException();
  };
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> throw new IllegalArgumentException();
  };
}

void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      throw new IllegalArgumentException();
  };
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> throw new IllegalArgumentException();
  };
}
```

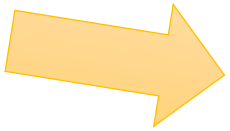
```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      throw new IllegalArgumentException();
  };
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      LOG.debug("Внезапно!");
      0.0;
  }
}
```

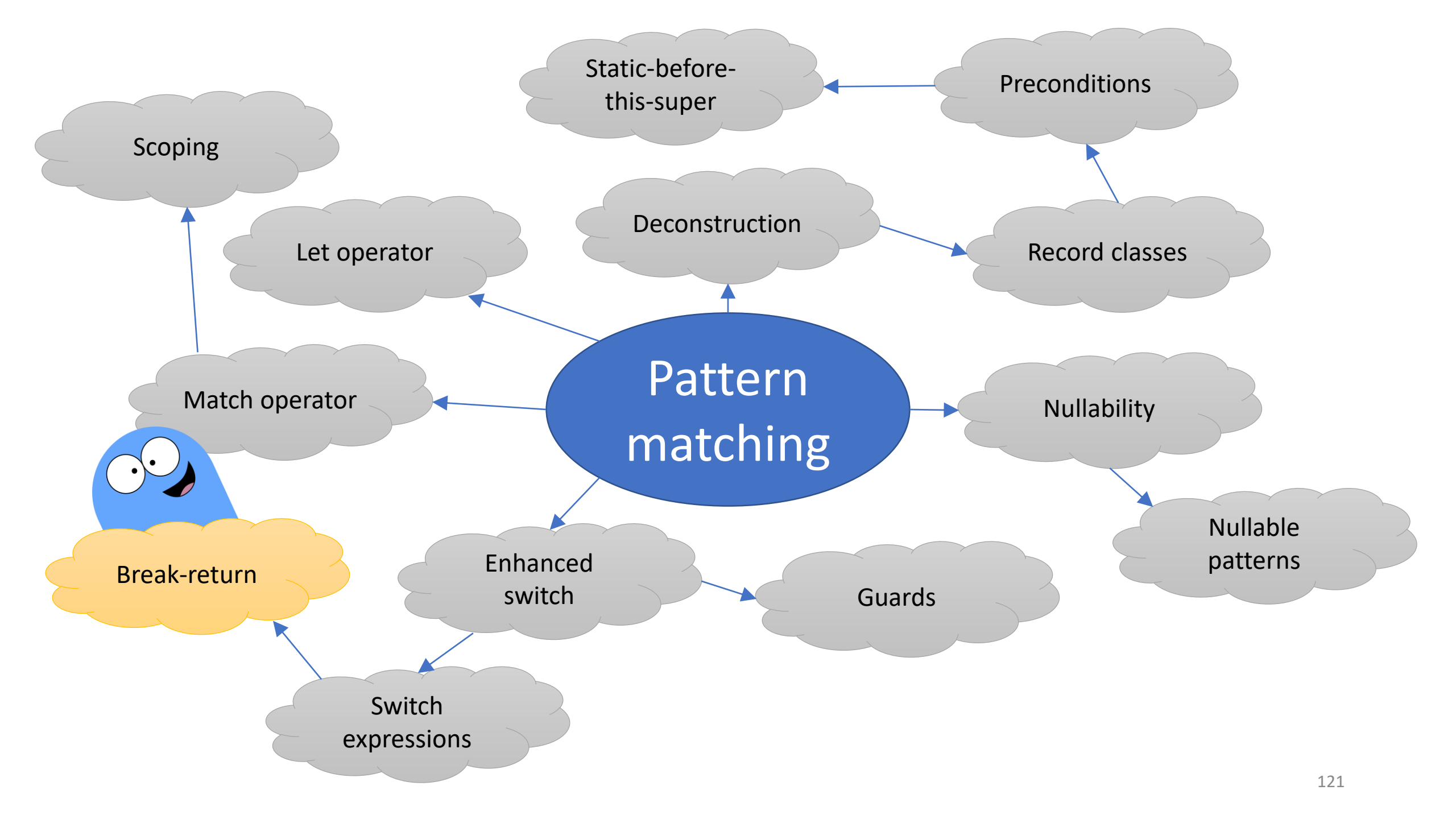
```
final Map<String, Integer> errorCounts = new HashMap<>();

void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      this.errorCounts.merge("Unexpected value", 1, Integer::sum);
      // А нолик написать и забыли :(
  }
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      LOG.debug("Внезапно!");
      return 0.0;
  }
}
```

```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      LOG.debug("Внезапно!");
      return 0.0;
  }
}
```



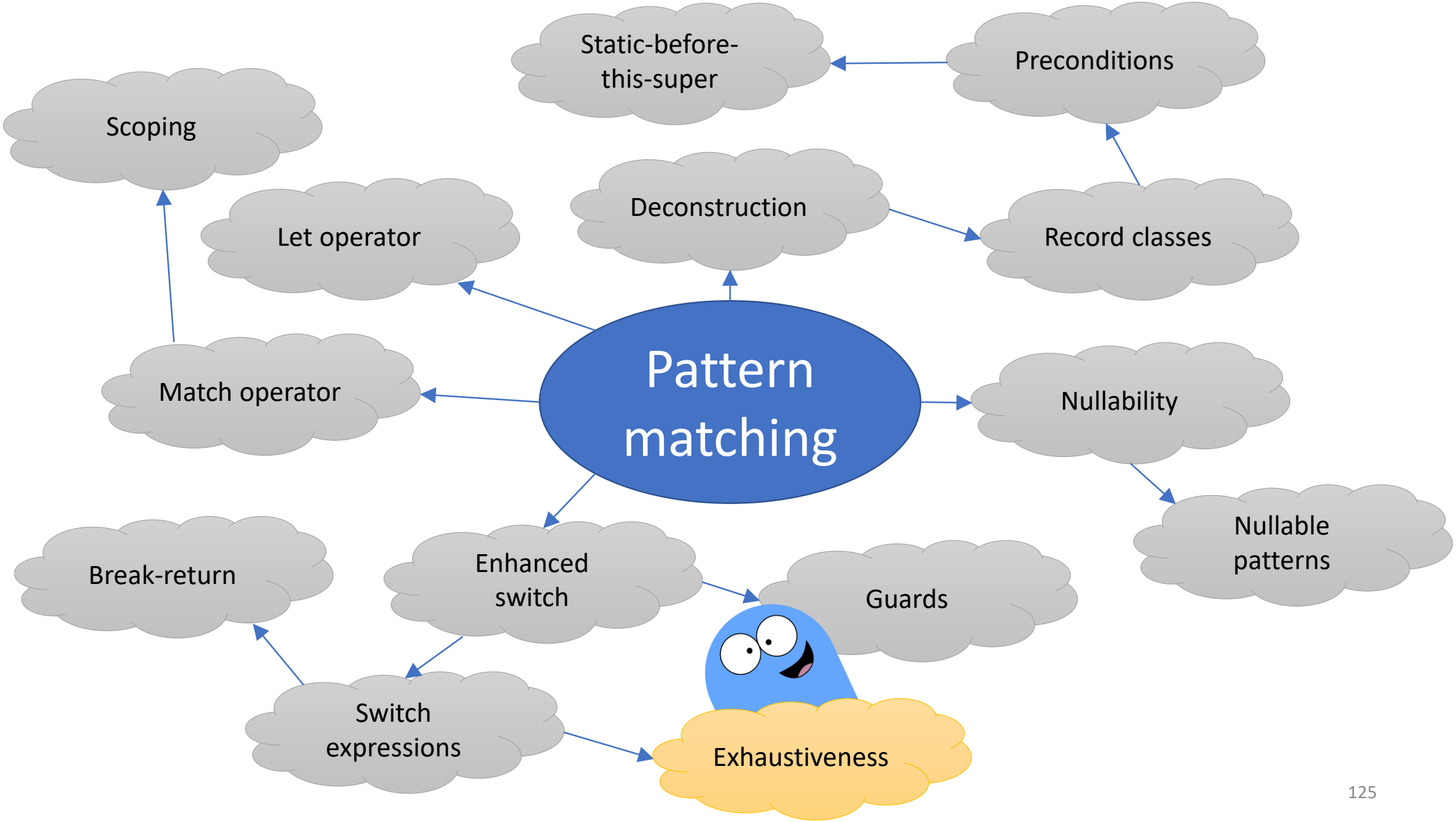
```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default -> {
      LOG.debug("Внезапно!");
      return 0.0;
    }
  }
}
```


Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Nullability
Break-return
Enhanced switch
Guards
Nullable patterns
Switch expressions

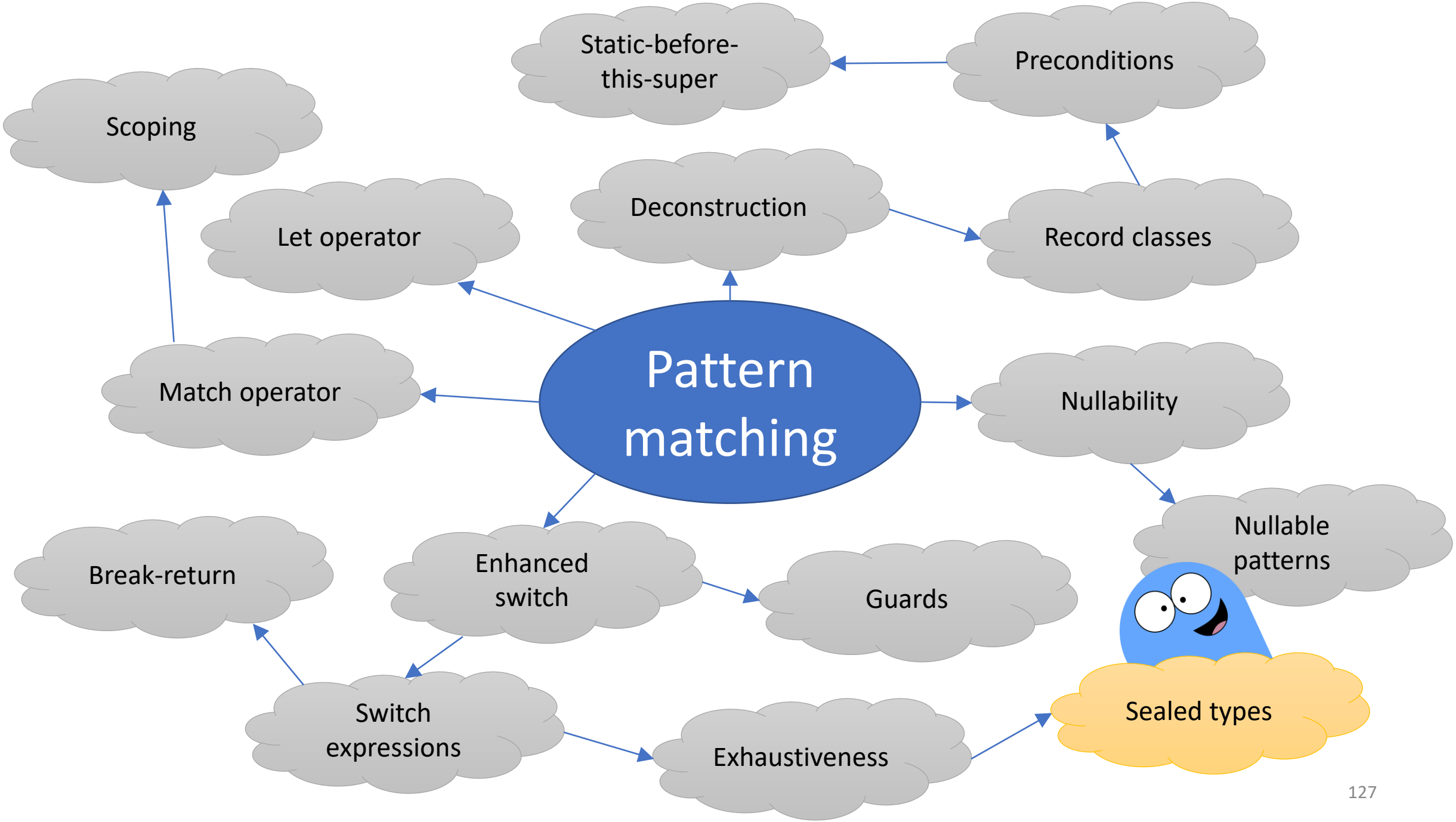
```
void test(Object obj) {
  Number n = switch (obj) {
    case Integer i -> i;
    case Double d -> d;
    default:
      LOG.debug("Внезапно!");
      break 0.0;
  }
}
```

```
OOPS_I_DID_IT_AGAIN:
for(int i=0; i<10; i++) {
  int OOPS_I_DID_IT_AGAIN = 5;

  System.out.println(switch(i) {
    case 1 -> 10;
    default:
      break OOPS_I_DID_IT_AGAIN;
  });
}
```

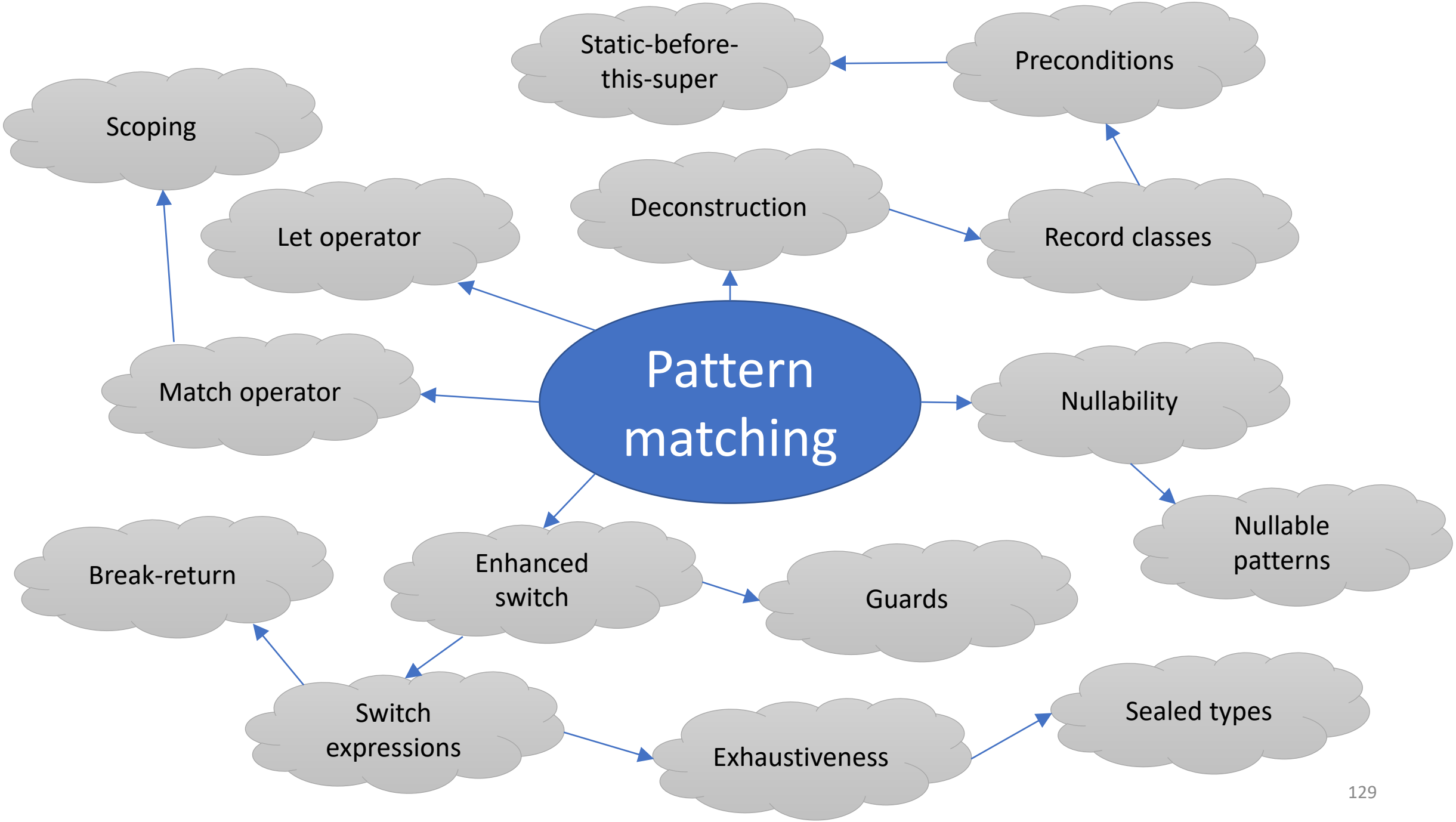
```
enum Size {
  L, M, S
}

String asString(Size size) {
  return switch(size) {
    case L -> "Large";
    case M -> "Middle";
    case S -> "Small";
  };
}
```


Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Nullability
Nullable patterns
Break-return
Enhanced switch
Guards
Switch expressions
Exhaustiveness

```
abstract class JavaConference {}
final class Joker extends JavaConference {}
final class JPoint extends JavaConference {}

String asString(JavaConference conf) {
  return switch(conf) {
    case Joker c  -> "Joker: "+c;
    case JPoint c -> "JPoint: "+c;
  };
}
```


Static-before-this-super
Preconditions
Scoping
Deconstruction
Let operator
Record classes
Pattern matching
Match operator
Nullability
Nullable patterns
Break-return
Enhanced switch
Guards
Switch expressions
Exhaustiveness
Sealed types

```
abstract sealed class JavaConference {
  class Joker extends JavaConference {}
  class JPoint extends JavaConference {}
}

String asString(JavaConference conf) {
  return switch(conf) {
    case Joker c  -> "Joker: "+c;
    case JPoint c -> "JPoint: "+c;
  };
}
```


Pattern matching
Scoping
Static-before-this-super
Preconditions
Let operator
Deconstruction
Record classes
Match operator
Nullability
Break-return
Enhanced switch
Guards
Nullable patterns
Switch expressions
Exhaustiveness
Sealed types

Что почитать:

JEP 305: Pattern Matching
http://openjdk.java.net/jeps/305

JEP 325: Switch Expressions (Preview)
http://openjdk.java.net/jeps/325

Data Classes for Java
http://cr.openjdk.java.net/~briangoetz/amber/datum.html

Pattern Matching for Java
http://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html

Pattern Matching for Java -- Semantics
http://cr.openjdk.java.net/~briangoetz/amber/pattern-semantics.html

Pattern Matching for Java -- Runtime and Translation
http://cr.openjdk.java.net/~briangoetz/amber/pattern-match-translation.html

The Amber Expert Group mailing list
http://mail.openjdk.java.net/pipermail/amber-spec-experts/

Спасибо
за внимание

—

https://twitter.com/tagir_valeev
https://habrahabr.ru/users/lany
https://github.com/amaembo
tagir.valeev@jetbrains.com