**Java и его приложения**

**1. Характеристики простых типов данных. Операции, выражения, правила приведения типов.**

Java имеет 8 простых (примитивных) типов данных: - Целочисленные: byte (8 бит), short (16 бит), int (32 бита), long (64 бита) - С плавающей точкой: float (32 бита), double (64 бита) - Символьный: char (16 бит) для символов Unicode - Логический: boolean (true/false)

Над этими типами можно выполнять различные операции: - Арифметические: сложение, вычитание, умножение, деление, остаток от деления - Сравнения: равно, не равно, больше, меньше - Логические операции для boolean значений - Побитовые операции для целочисленных типов

Правила приведения типов определяют, как преобразуются значения при смешивании разных типов: - Автоматическое приведение: меньший тип к большему (byte → int → long → float → double) - Явное приведение: требуется указание типа в скобках при возможной потере данных, например (int)3.14

double d = 100; // int автоматически преобразуется в double  
int x = (int)5.7; // double явно преобразуется в int, результат 5

**2. Операторы. Блок операторов. Управляющие операторы. Операторы перехода.**

**Блок операторов** – это группа операторов в фигурных скобках, рассматриваемая как единое целое.

Управляющие операторы меняют порядок выполнения программы: - if-else: выполняет код условно, в зависимости от логического выражения - switch: выбор из нескольких вариантов на основе значения выражения - Циклы: for (для заранее известного числа повторений), while (проверка условия до выполнения), do-while (гарантирует хотя бы одно выполнение)

Операторы перехода изменяют стандартный поток выполнения: - break: прерывает выполнение цикла или switch-case - continue: пропускает оставшуюся часть текущей итерации цикла - return: возвращает значение из метода и завершает его работу

if (x > 0) {  
 System.out.println("Положительное");  
} else {  
 System.out.println("Отрицательное или ноль");  
}  
  
for (int i = 0; i < 5; i++) {  
 if (i == 2) continue; // пропустит итерацию при i=2  
}

**3. Массивы в языке Java. Массив как параметр и тип возвращаемого значения метода. Аргументы метода main().**

**Массив** в Java – это объект, содержащий элементы одного типа, доступ к которым осуществляется по индексу (с нумерацией от 0). В отличие от коллекций, массивы имеют фиксированный размер после создания.

Объявление и создание массива выполняется двумя способами: 1. Раздельно: тип[] имя; имя = new тип[размер]; 2. Вместе с инициализацией: тип[] имя = {элементы};

Многомерные массивы создаются как массивы массивов, что позволяет каждой строке иметь разную длину.

При передаче массива в метод, передается ссылка на массив, поэтому изменения внутри метода видны вызывающему коду. Массив также может быть возвращаемым значением метода.

void incrementArray(int[] array) {  
 for (int i = 0; i < array.length; i++) {  
 array[i]++; // изменения видны снаружи метода  
 }  
}

Аргументы метода main() – это массив строк, содержащий параметры командной строки. Первый аргумент доступен как args[0], второй – args[1] и т.д.

**4. Классы в языке Java. Компоненты класса: данные и методы. Конструкторы. Ссылка this. Перегрузка методов. Final-компоненты. Статические компоненты класса. Операция «сборка мусора».**

**Класс в Java** – это шаблон для создания объектов, содержащий определение полей (данных) и методов (поведения).

Поля (переменные класса) хранят состояние объекта, а методы определяют операции, которые можно выполнять с объектом или над его данными.

**Конструкторы** – специальные методы, вызываемые при создании объекта (new) для его инициализации. Имя конструктора совпадает с именем класса, и он не имеет типа возвращаемого значения.

**Ссылка this** указывает на текущий объект и используется для: - разрешения конфликтов имен между полями и параметрами - вызова других конструкторов того же класса

Перегрузка методов позволяет определить несколько методов с одним именем, но разными параметрами, что улучшает читаемость кода.

**Final-компоненты**: - final переменные: константы, нельзя изменить после инициализации - final методы: нельзя переопределить в подклассах - final классы: нельзя наследовать

Статические компоненты принадлежат классу в целом, а не отдельным объектам: - статические поля: общие для всех объектов класса - статические методы: вызываются через имя класса, не имеют доступа к this

**Сборка мусора** – автоматический процесс освобождения памяти, занятой неиспользуемыми объектами. Точное время запуска сборщика мусора непредсказуемо.

**5. Наследование в Java. Суперкласс и подклассы. Конструкторы подкласса. Доступ к компонентам при наследовании. Переопределение методов.**

**Наследование** – механизм создания новых классов на основе существующих. Позволяет определить общее поведение в базовом классе и расширить его в производных классах.

Базовый класс называется суперклассом (родительским), а производный – подклассом (дочерним). Подкласс наследует поля и методы суперкласса, может добавлять новые и переопределять существующие.

Конструкторы не наследуются, но конструктор подкласса должен вызывать конструктор суперкласса (явно через super() или неявно). Вызов super() должен быть первым оператором в конструкторе.

Доступ к компонентам при наследовании зависит от модификаторов доступа: - private: доступны только в своем классе - default (без модификатора): в своем пакете - protected: в своем пакете и в подклассах - public: везде

Переопределение методов позволяет изменить поведение, унаследованное от суперкласса. При переопределении: - Сигнатура метода должна совпадать с сигнатурой в суперклассе - Рекомендуется использовать аннотацию @Override - Возвращаемый тип может быть подтипом исходного - Модификатор доступа не может быть более строгим

class Animal {  
 public void makeSound() {  
 System.out.println("...");  
 }  
}  
  
class Dog extends Animal {  
 @Override  
 public void makeSound() {  
 System.out.println("Гав!");  
 }  
}

**6. Абстрактные методы. Абстрактные классы и интерфейсы и их реализация.**

**Абстрактный метод** – метод без реализации (только объявление), который должен быть переопределен в подклассах. Объявляется с ключевым словом abstract.

Абстрактный класс содержит хотя бы один абстрактный метод и не может быть инстанцирован (нельзя создать объект). Служит как базовый класс для иерархии классов.

Абстрактный класс может содержать как абстрактные, так и конкретные методы, поля, конструкторы. Подкласс должен реализовать все абстрактные методы или сам быть абстрактным.

**Интерфейс** – полностью абстрактный тип, содержащий только абстрактные методы и константы. Задает контракт, который должны выполнять реализующие классы.

С Java 8 интерфейсы могут содержать: - Методы по умолчанию (default): с реализацией - Статические методы (static): принадлежат интерфейсу - С Java 9 добавлены приватные методы

Класс реализует интерфейс с помощью ключевого слова implements и должен предоставить реализацию всех его методов. Класс может реализовать несколько интерфейсов.

Основные отличия от абстрактных классов: - Множественная реализация: класс может реализовать несколько интерфейсов, но наследовать только один класс - Интерфейс не содержит состояние (только константы), абстрактный класс может иметь поля с состоянием

**7. Оболочки простых типов. Обзор пакета java.lang.**

Оболочки примитивных типов – классы, представляющие примитивные типы как объекты: - Byte, Short, Integer, Long, Float, Double, Character, Boolean

Эти классы позволяют использовать примитивные типы там, где требуются объекты (коллекции, обобщения), и предоставляют полезные методы для работы с ними.

Автоупаковка и автораспаковка (с Java 5) – автоматическое преобразование между примитивными типами и их объектными оболочками:

Integer obj = 10; // автоупаковка: int → Integer  
int num = obj; // автораспаковка: Integer → int

Полезные методы оболочек: - parseXxx() – преобразование строки в примитив - valueOf() – преобразование строки или примитива в объект-оболочку - xxxValue() – получение примитивного значения - MIN\_VALUE, MAX\_VALUE – константы с минимальным и максимальным значениями

Пакет java.lang автоматически импортируется во все программы Java и содержит наиболее фундаментальные классы: - Object – базовый класс для всех классов - String – неизменяемые строки - StringBuilder, StringBuffer – изменяемые строки - Math – математические функции - System – системные методы - Thread – многопоточность - Throwable – базовый класс для всех исключений

**8. Обработка исключительных ситуаций. Иерархия классов исключений. Создание собственных классов исключений.**

**Исключения в Java** – механизм обработки ошибок и необычных ситуаций во время выполнения программы.

Основной синтаксис обработки:

try {  
 // код, который может вызвать исключение  
} catch (TипИсключения e) {  
 // обработка исключения  
} finally {  
 // код, выполняемый всегда  
}

С Java 7 появился блок try-with-resources для автоматического закрытия ресурсов:

try (ресурсы автоматически закрываемые) {  
 // код, использующий ресурсы  
}

Иерархия исключений: - Throwable – корень иерархии - Error – серьезные ошибки, обычно не обрабатываются (OutOfMemoryError) - Exception – обрабатываемые исключения - Проверяемые (checked): IOException, SQLException - Непроверяемые (unchecked): RuntimeException и его подклассы

Проверяемые исключения должны быть либо обработаны (try-catch), либо объявлены в сигнатуре метода (throws). Непроверяемые исключения не требуют явной обработки.

Создание собственного класса исключения:

// Проверяемое исключение  
class MyException extends Exception {  
 public MyException(String message) {  
 super(message);  
 }  
}

Генерация исключения выполняется оператором throw:

if (value < 0) {  
 throw new IllegalArgumentException("Значение должно быть положительным");  
}