|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A Beginner’s Guide to Java: Part 1 of 4**  These blogs will help you learn Java Programming & Concepts in a simple and effective way.  If you have no prior knowledge in Java, you won’t face any difficulty. If you are experienced java developer, this blog will help you brush up the concepts.  **JVM ,JRE and JDK**  This is how the magic happens, you write your logic aka code in a java file, its converted into class file so that the machine can read your logic and run it.  https://cdn-images-1.medium.com/max/1000/1*dwYTp2UJxD0l_utvXEHy_A.png  JVM, JRE and JDK flow  Briefly these points covers it all:   * JVM is the java virtual machine that runs the java byte code. * JVM can be loaded on various hardware platforms, byte codes are the machine language of JVM. So Java is a better portable language. JVM is the entity that makes Java portable; there are different implementations of JVM for different OS (mac, windows, linux) etc. * JRE is java runtime environment that is sufficient to run the program. * JRE = JVM + library files/java package classes (Util, Lang, Math etc). * JDK is java development kit, required to write, compile and run a program. * JDK = JRE + Tools needed to develop java program.   **Memory Allocation**  So in the backgroud how the memory allcation works from your code. Brief pointers:   * Each time object is created in Java it is stored in heap memory. * Primitive variables and local are stored in stack, member variables in heap. * In multithreading each thread will have its own stack but will share same heap. We will discuss multithreading later in part 2. * Methods and variables are pushed to the stack when a method is invoked and stack pointer is decremented when call is completed. * 32 bit OS can’t use more than 4GB RAM for java application. 64 bit use more memory for same object, almost twice. * Primitive int uses 4 times less memory than Integer.   https://cdn-images-1.medium.com/max/1000/0*YnLwmAEJscRkHwx_.  Memory Allocation Representation  The below table gives an idea of various datatypes and range of values it can hold.  https://cdn-images-1.medium.com/max/1000/1*uOebRGa_vJm6x5xhtPF88w.png  Datatypes and Rage Allocation  **OOPS — Encapsulation, Inheritance, Polymorphism, and Abstraction**  Object Oriented Programming(OOP) is a programming concept that works on the 4 principles.  **1. Encapsulation**  Encapsulation is wrapping data(variables) and functionality(methods) together as a single unit. Functionalities mean “methods” and data means “variables”. Its all wrapped in is “class.” It is a blueprint or a set of instruction.  **Class**: A class is a blueprint or prototype that defines the variables and the methods. For example:  *Class: Car Data members or objects: color, type, model, etc. Methods: stop, accelerate, cruise.*  **Object**: Now, an object is a specimen of a class. Like in the above example my car is an object of the class Car.  **Variable**: can be local, instance and static. Local variables are declared inside the body of a method. Instance variables are declared outside method. They are object specific. Static variables are initialized only once and at the start of program execution. Static variables are initialized first, we will discuss static in detail later.  **Method**: methods are various functionalities, its nothing but set of code which is referred to by name and can be called (invoked) at any point in a program. You can pass multiple values to a method and it returns value(s).  **Package**: A Package is a collection of related classes. It helps organize classes into a folder structure and make it easy to locate and reuse them.  package com.example;  class Car {  String color = "black"; //instance variable   void accelerate() {   int speed = 90; //local variable  } }  **2. Abstraction**  Abstraction is selecting data from a larger pool to show only the relevant details to the object. Here is a chart showing different access modifiers and how it restricts the data from a class.  **3. Inheritance**  Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents.  class Developer{  public void writeCode(){  // writeCode method   } class BackendDeveloper extends Developer{  public void writeCode(){  // writeCode method  } } Class run{  public static void main (String args[]){  Developer developerObject = new Developer()  // writeCode method in class Developer will be executed  developerObject.writeCode();    BackendDeveloper backendDeveloperObj = new BackendDeveloper();  // writeCodemethod in class BackendDeveloper will be executed  backendDeveloperObj.writeCode();  } }  **4. Polymorphism**  Polymorphism is a OOPs concept where one name can have many forms also knows as overloading. Dynamic Polymorphism is the mechanism by which multiple methods can be defined with same name and signature in the superclass and subclass also known as overriding.   * Overloading is multiple methods in the same class with same name but different method signature. * Overriding deals with two methods, one in parent class and one in child class and both have same name and signature. * Subclass method overrides the method from super class. * In overriding sub classes access modifier must be greater than parent class E.g if we use *public abc()* in parent class and *private abc()* in sub class that will throw exception.   **Static Class Loading and Dynamic Class Loading**   * Loading the class to JVM to run is called class loading. * Classes are statically loaded using new operator. * Very first class is loaded using static main() method and then subsequent class are loaded. * Server based projects do not have main() at all, server provides infrastructure. Class to be loaded first is mentioned in config file. So the framework implements main() method and provides API in many cases. E.g: Container invokes init() method in servelets. * Main is required when Java program is run on JVM from command prompt. * NoClassDefinationFoundException is thrown if class reference is not found during static class loading. * Dynamic class loading is programmatically invoking class at run time. E.g: Class.forName(String ClassName); * ClassNotFoundException is thrown for dynamic class loading.   **Abstract Class and Interface**   * Interface has no implementation code and all methods are abstract i.e. all methods are only declared and none are defined. * Abstract class has executable methods and abstract methods. * A class can implement any number of interfaces but can extend only one abstract class. * In abstract class methods can be abstract and may not be. * An abstract class cannot be instantiated in can only be subclassed. * All abstract methods must be defined in subclass else the subclass should be abstract. * Interface cannot be instantiated it can only be implemented by other classes or extended by other interfaces. * Interface variables are final and static; interface methods are public and abstract by default. * Interface cannot contain implementation and cannot be subclassed, so variables have to be constant.   **Java Packages**  Here are some libraries available in java package to help code better. We will discuss them all eventually.  https://cdn-images-1.medium.com/max/1000/1*9-kl0KixCNNWIxo_6NmndA.png  Java Packages  **Constructor**   * The sole purpose of having Constructors is to create an instance of a class. They are invoked while creating an object of a class. * If a constructor with arguments has been defined in a class, you can no longer use a default no-argument constructor — you have to write one. * Java Doesn’t support Copy Constructor. * Constructor has same name as class. * Once constructor can be called from other using ‘this’ syntax, this means this object. * Java provides default constructor. * Private constructor: * Prevent class from being explicitly instantiated. * Object can be constructed but internally. * Used for singleton.   **Question**: Can constructors be synchronized in Java?  No. Java doesn’t allow multi thread access to object constructors so synchronization is not even needed.  **Question**: Are constructors inherited? Can a subclass call the parent’s class constructor?  You cannot inherit a constructor. By overriding a superclasses constructor you would erode the encapsulation abilities of the language. By Super keyword we can call the parents class contructor.  **Static**   * Static is used to have only one copy i.e. when we want to create variable or method that is shared by all objects of the class. * Static is used to share information across all the objects. * Static is used for variables, methods and block. * Static variables or method belong o the class and not to the object. * Static variable or method is initialized once before instance variable. * Static variable or method can be directly call from class name e.g. *<className>.<variableName>* * Static method can access only static data. * Static method cannot refer this or super. * Static method can only call other static methods. * Main method is static coz it must be accessible for an application to run before any instantiation takes place. * Constructor cannot be made static coz compiler will treat it as a method, also constructor is used to initialize new object where static is opposite of it. * Static variable is loaded first and then static block, although the sequence does matters. Static methods are loaded in the end. * Hirarchy is:   *Static parent → Static child → Instance parent → Constructor parent → Instance child → Constructor child.*   * While overriding static method, Complier doesn’t give any error and runs fine but it’s not overriding it is called **hiding**, coz we won’t get the benefits of run time polymorphism.   **Final, Finalize and Finally**   * Final keyword is used if we don’t want to change its value. * Final class cannot be extended. * Final method cannot be overridden. * Final variables are equivalent to constants. * Finally block is called in all cases for a try catch block, used to release system resources like connections, statements etc. We will discuss try, catch and finally blocks in detail. * Finalize() method helps garbage collection, this method is invoked before an object is discarded by garbage collector.   **Object Class**  Every class has Object as super class. It has the following non-final methods:   * equal() * hashCode() * toString() * clone() * finalize()   It has the following final methods:   * wait() * notify() * notifyAll() * getClass()   **Equals and HashCode**   * equals() and hashCode() methods are overridden to compare two objects. * equal() method makes equals comparison & hasCode method provide the hashCode.   public class Tiger {  private String color;  private String stripePattern;  private int height;  public String getColor() {  return color; }  public String getStripePattern() {  return stripePattern; }  public Tiger(String color, String stripePattern, int height) {  this.color = color;  this.stripePattern = stripePattern;  this.height = height;  }  @Override  public boolean equals(Object object) {  boolean result = false;  if (object == null || object.getClass() != getClass()) {  result = false;  } else {  Tiger tiger = (Tiger) object;  if (this.color == tiger.getColor() && this.stripePattern == tiger.getStripePattern()) {  result = true;  }   }  return result;  }  @Override  public int hashCode() {  int hash = 3;  hash = 7 \* hash + this.color.hashCode();  hash = 7 \* hash + this.stripePattern.hashCode();  return hash;  } }  **Clone**   * Clone method is used to copy an object. * Clone method has protected access modifier. * To call the clone method the object must implement Clonable interface, else it will throw CloneNotSupportedException. * Clonable interface is markers interface i.e. no methods defined interface. They just tell the class that it needs to be treated differently. * The advantage of having clonable is we can clone only those objects that allow us to clone. * If any field of an object is referred in other object, we will use shallow copy. In shallow copy only the memory address is copied i.e. same object is shared. * In deep copy the object is created and dynamically new memory is allocated.   Public Object Clone(){  Try{  Return super.clone();  }}  Public Object Clone(){  Try{  Object obj = (Object) super.clone();  Return obj;  }}  Don’t worry about the try statement, we will discuss in detail eventually.  **Aggregation and composition**   * Aggregation expresses an ‘IS A’ relationship. E.g: House is a building. * Composition expresses ‘HAS A’ relationship. E.g: House has a bathroom. This is a part of whole relationship where a part cannot exist without the whole. * Aggregation is a weaker relationship and composition is stronger. * Aggregations are generally achieved by extending a class and composition by implementing interface.   **Primitive and Wrapper Type**  A variable of a primitive type directly contains the value of that type. Java has eight primitive types: byte, short, int, long, char, boolean, float and double.  A **Wrapper class** is a **class** whose object wraps or contains a **primitive** data types. When we create an object to a **wrapper class**, it contains a field and in this field, we can store a **primitive** data types and various other supporting, operational methods. It *is* slower to use the Object wrappers for primitives than just using the primitives. You’re adding the cost of object instantiation, method calls, etc. Each of Java’s eight primitive data types has a class dedicated to it like Byte, Short, Integer, Long, String, Boolean, Float and Double.  **Autoboxing and Unboxing**   * Java 1.5 compiler provides automatic conversion of primitive datatype to wrapper type, this is known as Autoboxing and the reverse is Unboxing. * Compiler internally uses valueOf() and intValue() for the same.   **Casting**   * Assigning a value to other primitive value is casting.   *byte → short → int → long → float → double*   * Upcasting is possible, e.g:   int i = 5; long j = i;   * Downcasting is not possible, needs explicit casting:   long j = 5;  int i = j; (THIS IS WRONG, it will give classCastException)  int i = (int) j;   * int to String casting is not possible. | **Java для начинающих. Часть 1 из 4**  Данные статьи помогут легко и быстро разобраться концепциях и программировании на Java.  Даже при нулевых знаниях о Java трудностей в освоении этих материалов не возникнет. А опытные Java-разработчики смогут освежить свои знания.  **JVM ,JRE и JDK**  Вот так и происходит вся магия: логика (т.е. код) прописывается в java файле, который затем преобразуется в файл класса. Машина его читает и выполняет.  https://cdn-images-1.medium.com/max/1000/1*dwYTp2UJxD0l_utvXEHy_A.png  JVM, JRE и JDK поток  Это краткое изложение следующего:   * JVM – виртуальная Java машина, выполняющая байт-код Java. * JVM можно загружать на разном железе. Байт-коды – это машинный язык JVM. Поэтому Java является самым переносимым языком. JVM – это некий объект, который обеспечивает высокую переносимость. Для разных операционных систем (Mac, Windows, Linux) придуманы свои реализации JVM. * JRE – среда выполнения Java, достаточная для запуска программы. * JRE = JVM + файлы библиотеки/пакеты классов (Util, Lang, Math etc). * JDK – пакет средств разработки на Java. Нужен для написания, компиляции и выполнения программы. * JDK = JRE + инструменты, необходимые для разработки Java-программы.   **Выделение памяти**  Коротко о главном: как происходит выделение памяти со стороны кода в фоновом режиме.   * Каждый раз при создании объекта в Java он сохраняется в динамической памяти. * Примитивы и локальные переменные хранятся в стеке, переменные-члены – в области кучи. * При многопоточности каждый поток имеет собственный стек, но находится в общей куче. О многопоточности поговорим в части 2. * При вызове метода все методы и переменные помещаются в стек. По завершении вызова счетчик стека уменьшается. * 32-разрядная операционка не тратит более 4 ГБ RAM на Java-приложения. В 64-разрядной затраты памяти на те же элементы увеличиваются вдвое. * Примитивный тип int тратит в 4 раза меньше памяти, чем Integer.   https://cdn-images-1.medium.com/max/1000/0*YnLwmAEJscRkHwx_.  Графическое представление распределения памяти  Таблица ниже перечисляет различные типы данных и диапазоны хранимых значений.  https://cdn-images-1.medium.com/max/1000/1*uOebRGa_vJm6x5xhtPF88w.png  Типы данных и диапазоны значений  **ООП —  Инкапсуляция, наследование, полиморфизм и абстракция**  Объектно-ориентированное программирование (ООП) – это концепция программирования, основанная на 4 базовых принципах.  **1. Инкапсуляция**  Инкапсуляция – это объединение данных и функциональных средств в единый компонент. Функциональные средства – это «методы», а данные – это «переменные». Все они оборачиваются в «класс». Это некая схема или набор инструкций.  **Класс**: Класс – это некий прообраз или прототип, который определяет переменные и методы. Пример:  *Класс: Машина Переменные-члены или объекты: цвет, тип, модель и т.д.*  *Методы: остановка, ускорение, предельная скорость.*  **Объект:** Объект – это экземпляр класса. В примере выше моя машина будет экземпляром общего класса Машина.  **Переменные**: локальные, статические и переменные экземпляра. Локальные переменные объявляются в теле метода. Переменные экземпляра объявляются вне метода, являются специфическими для объекта. Статические переменные инициализируются только один раз при запуске программы. Статические переменные инициализируются первыми, но об этом чуть позже.  **Метод**: Методы – это различные функциональные средства. То есть, не что иное, как набор кода, на который поименно ссылаются или вызывают из любой части программы. Вы передаете несколько значений в метод, а он их (эти значения) возвращает.  **Пакет**: Пакет – это набор связанных классов. Используется для организации классов в структуру папок, а также для быстрого нахождения и повторного использования этих классов.  package com.example;  class Car {  String color = "black"; //instance variable   void accelerate() {   int speed = 90; //local variable  } }  **2. Абстракция**  Абстракция – это выбор данных из большого пула по определенным критериям объекта. Ниже показаны различные модификаторы доступа и то, как они ограничивают данные из класса.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Модификатор доступа** | **Тот же пакет** | | | **Разный пакет** | | | **Класс** | **Подкласс** | **Другой класс** | **Подкласс** | **Другой класс** | | без модификатора | ДА | ДА | ДА |  |  | | public | ДА | ДА | ДА | ДА | ДА | | protected | ДА | ДА | ДА | ДА |  | | private | ДА |  |  |  |  |   **3. Наследование**  Наследование – это процедура, при которой один класс приобретает свойства другого. Например, потомок наследует признаки своего родителя.  class Developer{  public void writeCode(){  // writeCode method   } class BackendDeveloper extends Developer{  public void writeCode(){  // writeCode method  } } Class run{  public static void main (String args[]){  Developer developerObject = new Developer()  // writeCode method in class Developer will be executed  developerObject.writeCode();    BackendDeveloper backendDeveloperObj = new BackendDeveloper();  // writeCodemethod in class BackendDeveloper will be executed  backendDeveloperObj.writeCode();  } }  **4. Полиморфизм**  Полиморфизм – это концепция ООП, при которой одно имя принимает множество форм, (другое название – перегрузка). Динамический полиморфизм – это механизм, при котором несколько методов определяются одним именем и сигнатурой подкласса или суперкласса (другое название – переопределение).   * Перегрузка – это несколько методов одного класса с одним именем, но разной сигнатурой. * Переопределение – два метода (один в родительском классе, другой – в дочернем) с одним именем и сигнатурой. * Метод подкласса переопределяет метод суперкласса. * При переопределении подклассов модификатор доступа должен быть больше родительского класса. Например, *public abc()* в родительском классе и *private abc()* в подклассе выбрасывает исключение.   **Загрузка статического и динамического класса**   * Добавление класса для запуска в JVM называется загрузкой класса. * Классы загружаются статично с помощью нового оператора. * Первый класс загружается через метод static main(). Затем подгружаются остальные классы. * В серверных проектах отсутствует main(), поскольку сервер сам отвечает за всю инфраструктуру. Первый класс для загрузки отмечается в config файле. Довольно часто фреймворк реализует метод main() и предоставляет API. Пример: Контейнерный класс вызывает метод init() в сервлетах. * main нужен для запуска Java-программы из командной строки в JVM. * Если при загрузке статического класса не находится ссылка на класс, то выбрасывается NoClassDefinationFoundException. * Динамические классы загружаются через программный вызов при выполнении. Пример: Class.forName(String ClassName); * ClassNotFoundException выбрасывается при загрузке динамического класса.   **Абстрактный класс и интерфейс**   * В интерфейсе отсутствует код реализации, а все методы являются абстрактными. То есть, все методы объявляются, но ни один не определяется. * В абстрактном классе есть исполняемые и абстрактные методы. * Класс реализует сколько угодно интерфейсов, но расширяет только один абстрактный класс. * Методы абстрактного класса могут быть или не быть абстрактными. * Абстрактный класс не может превратиться в экземпляр, но может стать подклассом. * Все абстрактные методы должны определяться в подклассе, то есть, подкласс является абстрактным. * Создавать экземпляры из интерфейса нельзя. Их можно реализовывать в других классах или расширять другими интерфейсами. * Переменные интерфейсов конечные и статичные. По умолчанию, все методы интерфейса публичные и абстрактные. * Интерфейс не может содержать реализацию и не может превращаться в подкласс. Все переменные должны быть постоянными.   **Пакеты Java**  Ниже даны примеры библиотек из пакета Java для правильного написания кода. О них мы еще поговорим.  https://cdn-images-1.medium.com/max/1000/1*9-kl0KixCNNWIxo_6NmndA.png  Пакеты Java  **Конструкторы**   * Их единственная цель – создавать экземпляры класса. Они вызываются в процессе создания объекта класса. * Если конструктор с аргументами определен в классе, то нельзя будет работать со стандартным конструктором без аргументов (no-argument constructor) – придется их прописать. * Java не поддерживает конструктор копирования. * Имя конструктора и класса совпадает. * Если конструктор вызывается из другого конструктора синтаксисом с this, то речь идет именно об этом объекте. * В Java есть стандартный конструктор. * Приватный конструктор: * Защищает класс от явного превращения в экземпляр. * Построение объекта возможно только внутри конструктора. * Используется в шаблоне «Одиночка» (Singleton).   **Вопрос**: Можно ли синхронизировать конструкторы в Java?  Нет. В Java запрещен многопоточный доступ к конструкторам объекта, поэтому необходимость в синхронизации отсутствует.  **Вопрос**: Наследуются ли конструкторы? Может ли подкласс вызывать конструктор родительского класса?  Конструкторы не наследуются. При переопределении конструктора суперклассов нарушается инкапсуляция языка. Конструктор родительского класса вызывается ключевым словом super.  **Static**   * Static используется для обозначения чего-то в единственном экземпляре. Пример: в методе или переменной, доступных для всех объектов класса. * Static необходим для передачи информации по всем объектам. * Static подходит для переменных, методов и блоков. * Статичный метод или переменные принадлежат классу, а не объекту. * Статичный метод или переменная инициализируются до переменной экземпляра. * Статичный метод или переменная могут вызываться напрямую из имени класса. Пример: *<className>.<variableName>* * Статичный метод имеет доступ только к статичным данным. * Статичный метод не может ссылаться на this или super. * Статичный метод может вызывать только другие статичные методы. * main () – это статичный метод. Он должен быть доступен приложению до создания экземпляров. * Конструктор не бывает статичным, потому как компилятор считает его методом. Кроме того, конструктор нужен для инициализации нового объекта, а static выполняет совершенно противоположную функцию. * Статичная переменная загружается первой. После нее идет статичный блок. И очередность здесь важна. Статичные методы загружаются в конце. * Иерархия следующая:   *Статичный родитель → Статичный потомок → Экземпляр родитель → Конструктор родитель → Экземпляр потомок → Конструктор потомок.*   * При переопределении статичного метода компилятор не выдает ошибок. Но правильнее это называть не переопределением, а **скрытием,** т.к. пропадают все плюсы полиморфизма.   **Final, Finalize и Finally**   * Ключевое слово final указывает на неизменность значения чего-либо. * Класс final не расширяется. * Метод final не переопределяется. * Переменные final равнозначны константам. * Блок finally вызывается для всех блоков try-catch и используется для очистки системных ресурсов, будь то подключения, выражения и т.д. Мы еще поговорим о них подробнее. * Метод finalize() помогает высвобождать память. Он вызывается перед тем, как сборщик мусора помещает объект на удаление.   **Класс Object**  В каждом классе есть суперкласс Object. В нем присутствуют следующие не конечные методы:   * equal() * hashCode() * toString() * clone() * finalize()   Конечные методы суперкласса:   * wait() * notify() * notifyAll() * getClass()   **Equals и hashСode**   * Методы equals() и hashСode() переопределяются для сравнения двух объектов. * equal() выполняет сравнение, а hashcode возвращает хеш-код.   public class Tiger {  private String color;  private String stripePattern;  private int height;  public String getColor() {  return color; }  public String getStripePattern() {  return stripePattern; }  public Tiger(String color, String stripePattern, int height) {  this.color = color;  this.stripePattern = stripePattern;  this.height = height;  }  @Override  public boolean equals(Object object) {  boolean result = false;  if (object == null || object.getClass() != getClass()) {  result = false;  } else {  Tiger = (Tiger) object;  if (this.color == tiger.getColor() && this.stripePattern == tiger.getStripePattern()) {  result = true;  }   }  return result;  }  @Override  public int hashCode() {  int hash = 3;  hash = 7 \* hash + this.color.hashCode();  hash = 7 \* hash + this.stripePattern.hashCode();  return hash;  } }  **Clone**   * Метод сlone нужен для копирования объекта. * В методе clone присутствует защищенный модификатор доступа. * Для вызова метода clone объекту требуется реализация интерфейса Cloneable. В противном случае выбрасывается исключение CloneNotSupportedException. * Интерфейс Cloneable является маркерным, то есть методы не определяют интерфейс, а говорят классу об особом отношении. * Плюс такого интерфейса: можно копировать только объекты, доступные для клонирования. * Если какое-то поле объекта ссылается на другой объект, то делаем поверхностную копию. В ней копируется только адрес памяти, т.е. используется один и тот же объект. * При глубоком копировании происходит создание объекта и новое динамическое распределение памяти.   Public Object Clone(){  Try{  Return super.clone();  }}  Public Object Clone(){  Try{  Object obj = (Object) super.clone();  Return obj;  }}  Не обращайте внимание на оператора try – к нему мы вернемся позже.  **Агрегация и композиция**   * Агрегация выражает отношение *is a* («являться чем-то»). Пример: дом является зданием. * Композиция выражает отношение *has a* («быть частью чего-то»). Пример: в доме имеется ванная. То есть форма отношения, в котором эта часть не может существовать без остальных элементов. * Агрегация – это более слабое отношение. Композиция намного сильнее. * Как правило, агрегация достигается расширением класса. Для композиции нужна реализация интерфейса.   **Примитивы и оболочки типов**  Переменная примитивного типа всегда содержит его значение. В Java существует 8 примитивных типов: byte, short, int, long, char, boolean, float и double.  **Класс-оболочка** – это **класс,** объект которого оборачивает или содержит **примитивные** типы данных. При создании объекта в **классе-оболочке** появляется поле для хранения **примитивных** типов данных, а также других поддерживающих и операционных методов. Если использовать не сами примитивы, а Object-оболочки для примитивных типов данных, то процесс выполняется *медленнее.* Дополнительные ресурсы тратятся на создание экземпляра объекта, вызовы методов и т.д. За каждым из этих примитивных типов закреплен свой класс: Byte, Short, Integer, Long, String, Boolean, Float и Double.  **Автоупаковка и распаковка**   * Компилятор Java 1.5 автоматически преобразует примитивы к оболочкам типов, то есть выполняет автоупаковку. Обратное действие называется распаковкой. * Для этого в компиляторе используется valueOf() и intValue().   **Кастинг**   * Это присвоение значение другому примитиву.   *byte → short → int → long → float → double*   * В Java возможно и восходящее преобразование (upcasting):   int i = 5; long j = i;   * Нисходящее преобразование (downcasting) невозможно, т.к. требуется прямой кастинг:   long j = 5;  int i = j; (THIS IS WRONG, it will give classCastException)  int i = (int) j;   * Преобразование int в String также невозможно. |

https://medium.com/@madhupathy/a-beginners-guide-to-java-part-1-of-3-33edf47e47b4