Оглавление

[Mahat 2](#_Toc204344122)

[M2025.A4 2](#_Toc204344123)

[Car. Motorcycle – hard array OOP. 2](#_Toc204344124)

[S10\_Врачи – big MAAKAV 4](#_Toc204344125)

[B10\_Coach – big MAAKAV 6](#_Toc204344126)

[M2025\_KA 9](#_Toc204344127)

[S4. Cars – приведения типов. 9](#_Toc204344128)

[Ebook. Интерфейсы 10](#_Toc204344129)

[S10\_Подъебка. AA AB 12](#_Toc204344130)

[M2021\_A 13](#_Toc204344131)

[S1\_Tablet Store – simple array oop 13](#_Toc204344132)

[S3\_DressCompany – simple arr erusha 15](#_Toc204344133)

[S4\_A-B – «если-бы да-кабы» 15](#_Toc204344134)

[S5\_A-B – hard ERUSHA. 17](#_Toc204344135)

[S6\_Banana – hard EQUALS. 18](#_Toc204344136)

[EKZAMEN 19](#_Toc204344137)

[Буквы, erusha, если бы да кабы. 19](#_Toc204344138)

[A B. Маакав 24](#_Toc204344139)

# Mahat

# M2025.A4

## Car. Motorcycle – hard array OOP.

Есть 2 класса. Car и Motorcycle.

|  |  |
| --- | --- |
| public class Car {  **private** int speed;  public Car(int s){ speed = s; }  public int getSpeed(){ return speed;}  public boolean **equals**(Car other){  return (other != null) &&  (speed == other.speed);  }} | public class Motorcycle {  **private** int speed;  public Motorcycle(int s){ speed = s; }  public int getSpeed(){ return speed; }  public boolean **equals**(Object other){  return (  (other != null) &&  (other instanceof Motorcycle) &&  (speed == ((Motorcycle)other).speed)  ); }} |
| Car c1 = new Car (180); Object c2 = new Car (180); Motorcycle m1 = new Motorcycle (180); Object m2 = new Motorcycle (180);  Object[] vehicles = {  c1, c2,m1, m2,  new Motorcycle(200), new Motorcycle(220), new Motorcycle(260)}; | |

A) Напишите функцию, которая получает массив ссылок на объекты типа Object. A) Функция вернет количество мотоциклов (объекты типа Motorcycle), чья максимальная скорость находится между 180 и 250 км/ч.

|  |  |
| --- | --- |
| public static int countCars(Object[] v){  int count = 0;  Motorcycle m;  **for** (int i=0; i<v.**length**; i++){  **if** (v[i] instanceof Motorcycle) {  m = (Motorcycle) v[i];  **if** (m.getSpeed() >= 180 && m.getSpeed() <= 250)  count++; } }  return count;} | Здесь у нас DownCasting от Object -> Motorcycle  m = (Motorcycle) v[i]; |

Б) напиши механизм, и если есть ошибка – то какая именно.

Подьебка в том, что методы сравнения в этих исходных классах немного разные.

|  |  |
| --- | --- |
| Car c1 = new Car (180);  Object c2 = new Car (180);  Motorcycle m1 = new Motorcycle (180);  Object m2 = new Motorcycle (180); |  |
| 11) Syso(c1.speed); | לא תקין. שגיאת קומפילציה. תכונה "ספייד" מוגדרת בהרשהת גישה "פריבט" ולכן לא ניתן לגשת אליה מחוץ מחלקה "קאר"  Если изменить в Car с pivate на public поле speed - то скомпилится. |
| 12) Syso(((Motorcycle)c2).getSpeed() ); | לא תקין. שגיאת ריצה. המרה מטה מפורשת, אבל לא לצורך שחזור. אין השר ירושה בין "קאר" ל "מוטו"  Явное приведение вниз по иерархии, но не для возврата/восстановления исходного типа. |
| 13) Syso( c1.**equals**(c2) ) | **תקין.** תמומש הפעולה **equals** של Object  )כי הפרמטר מטיפוס . result = False. (c2 Object  Результат сравнения будет False. Здесь нет никакого OverRiding. |
| 14) Syso( c2.**equals**(c1) ) | תקין. upCasting. תמומש הפעולה **equals** של Object (כי אין דריבה של הפעולה **equals** של Object ב (Car  יחדיר False |
| 15) Syso( m1.**equals**(m2) ) | תקין. True.  תמומש הפעולה **equals** ב Moto |
| 16) Syso( m2.**equals**(m1) ) | תקין. True.  OverRiding and upCasting |
| 17) Syso( c1.**equals**((Motorcycle)m2)) | תקין. יחזיר False. תמומש הפעולה **equals** של Object  (upcasting של הפרמטר מ Motor ל (Object  Тут upCasting мотоцикла в Object. Поэтому нет ошибки чтения. Но в Car сравнивают только Car и Car. Результат False |
| 18) Syso ( c1.**equals**((Car)c2)) | תקין. מחזיר True    Тут успешный даункастинг объекта в Car. True |
| 19) Syso( m2.**equals**((Car)c2) ) | Тут даун кастинг Object с2 в Car. И последующий upCasting обратно в Object. В **equals** OverRide.  Вернет False. |
| 20) Syso( m1.**equals**((Motorcycle)c2) ) | Ошибка. Нельзя привести машину к мотоциклу.  Ошибка исполнения. Не компиляции.  לא תקין – שגיאהת ריצה. המרה מטה מפורשת. אבל לא לצורך שחזור. אין קשר ירושנ בין Car ל Motor |

## S10\_Врачи – big MAAKAV

|  |  |  |
| --- | --- | --- |
| public class Doctor {  protected String name;  protected String specialization;  protected int numOfPatients;   public Doctor(String name, String spec) {  this.name = name;  this.specialization = spec;  this.numOfPatients = 0;  }   public Doctor(String name, String spec, int num) {  this.name = name;  this.specialization = spec;  this.numOfPatients = num;  } | public Doctor(Doctor other) {  this.name = other.name;  this.specialization = other.specialization;  this.numOfPatients = other.numOfPatients;  }  set{}, get{}…  public String toString() {  return "Doctor:" +name+ ", "+ specialization+ ", "+ numOfPatients;  } | |
| public void addPatients(int num){  **if** (numOfPatients+num >=0)  this.numOfPatients+=num; } |
| public class Intern extends Doctor {  **private** Doctor mentor;  public Intern(String name, String spec, Doctor mentor){  super(name, spec);  this.mentor = mentor;  this.numOfPatients = mentor.numOfPatients/2;  }  public Doctor getMentor(){  return mentor;  }   public String toString() {  return "Intern: " + name + ", " + specialization +  ", Mentor: " + mentor.name +", " +  mentor.numOfPatients +"," + numOfPatients;  } | |  |

А B)Задание: исходя из этого кода распиши что будет до изменения, и что – после. Сделай таблицу маакав.

Doctor[] d = new Doctor[9];  
d[0] = new Doctor("Dr. Cohen", "Cardiology",12);

*//* Dr. Cohen, Cardiology, 12  
d[1] = new Doctor("Dr. Levy", "Neurology");

*//* Dr. Levy, Neurology, 0  
d[2] = new Doctor("Dr. Sharon", "Pediatrics",8);

*//* Dr. Sharon, Pediatrics, 8  
d[3] = new Intern("Dani", "Cardiology", new Doctor(d[0]));

*//* Int Dani, Cardiology, Mentor: Dr. Cohen, 12,6  
d[4] = new Intern("Yael", "Surgery", d[0]);

*//* Intern: Yael, Surgery, Mentor: Dr. Cohen, 12,6  
d[5] = new Intern("Avi", "Pediatrics", new Doctor(d[2]));

*//* Intern: Avi, Pediatrics, Mentor: Dr. Sharon, 8,4  
d[6] = new Intern("Ruth", "Oncology", d[2]);  
*//* Intern: Ruth, Oncology, Mentor: Dr. Sharon, 8,4

d[7] = new Intern("Noam", "Cardiology", new Doctor(d[1]));

*//* Intern: Noam, Cardiology, Mentor: Dr. Levy, 0,0  
d[8] = new Intern("Maya", "Neurology", new Doctor(d[0]));

*//* Intern: Maya, Neurology, Mentor: Dr. Cohen, 12,6

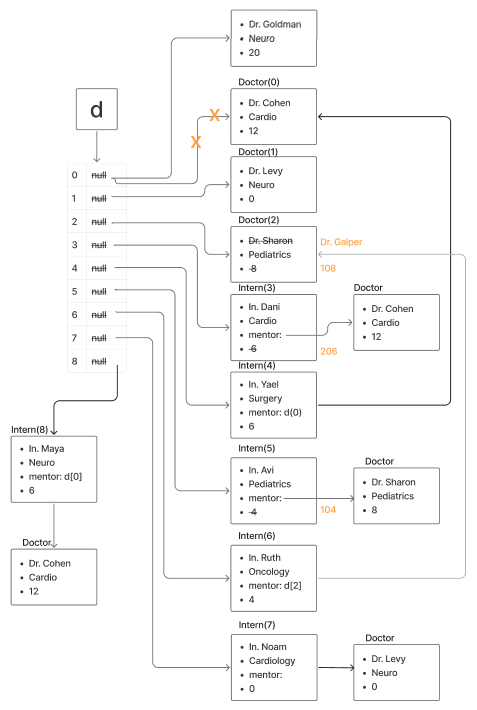
До: **for** (int i=0; i<d.**length**; i++) { System.out.println(i+" \_ "+d [i]);}

После:

d[0] = new Doctor("Dr. Goldman","Neurology",20);  
d[2].setName("Dr. Galper");  
d[2].addPatients(100);  
d[3].addPatients(200);  
d[5].addPatients(100);

**for** (int i=0; i<d.**length**; i++) { System.out.println(i+" \_ "+d[i]);}

Подъебка - после изменений в объекте, которого “вбирает” в себя интерн -  
зависимые от него параметры не изменяются по умолчанию и остаются старые.  
Чтобы этого не было, изменения надо прописывать через сеттеры



0 \_ Doctor:Dr. Goldman, Neurology, 20

1 \_ Doctor:Dr. Levy, Neurology, 0

2 \_ Doctor:Dr. Galper, Pediatrics, 108

3 \_ Intern: Dani, Cardiology, Mentor: Dr. Cohen, 12,206

4 \_ Intern: Yael, Surgery, Mentor: Dr. Cohen, 12,6

5 \_ Intern: Avi, Pediatrics, Mentor: Dr. Sharon, 8,104

6 \_ Intern: Ruth, Oncology, Mentor: Dr. Galper, 108,4

7 \_ Intern: Noam, Cardiology, Mentor: Dr. Levy, 0,0

8 \_ Intern: Maya, Neurology, Mentor: Dr. Cohen, 12,6

C) Напиши функцию, которая принимает массив из врачей/интернов и печатает имена интернов, специализация которых не совпадает со специализацией менторов.

public static void printNotSameSpec(Doctor[] d){

**for** (int i = 0; i<d.**length**; i++){

**if** ( d[i] instanceof Intern){  
 Intern intern;  
 intern = ((Intern) d[i]);

}

If(d[i].getSpecialization().**compareTo**(intern.getMentor().getSpecialization()) !=0)  
 System.out.println(d[i].getName()); }}}

## B10\_Coach – big MAAKAV

|  |  |  |
| --- | --- | --- |
| public class Coach {  protected int num;  protected int sessions;  protected String specialty;  public Coach(int num, int sessions, String specialty) {  this.num = num;  this.sessions = sessions;  this.specialty = specialty;  }  public void setNum(int n) {  this.num = n; } | public Coach(Coach other) {  this.num = other.num;  this.sessions = other.sessions;  this.specialty = other.specialty; }  @Override public String toString() {  return "Coach: " + this.num + ", sessions: " + this.sessions + ", specialty: " + this.specialty; } | |
| public void reduceSessions(int s) {  if (s > 0) {  this.sessions = Math.max(0, this.sessions - s);  }} |
| public class Athlete extends Coach {  **private** Coach personalCoach;   public Athlete(int n, int sessions, Coach personalCoach) {  super(n, sessions, "Athlete");  this.personalCoach = personalCoach;  this.sessions = Math.max(0, personalCoach.sessions);  }  public String toString() {  return "Athlete: " + num + ", sessions: " + sessions +  ", Coach: " + personalCoach.num +  ", sessions: " + personalCoach.sessions +  ", specialty: " + personalCoach.specialty;  }} | |  |

А B)Задание: исходя из этого кода распиши что будет до изменения, и что – после. Сделай таблицу маакав.

|  |  |
| --- | --- |
| Coach[] c = new Coach[4]; c[0] = new Coach(123, 12, "Jym"); c[1] = new Coach(234, 8, "CrossFit"); c[2] = new Athlete(345, 6, new Coach(c[0])); c[3] = new Athlete(456, 10, c[1]);  System.*out*.println("\nOld Team:"); **for** (int i = 0; i < c.length; i++) {  System.*out*.println(c[i]); }  c[0] = new Coach(567, 8, "Studio"); c[1].setNum(678); c[2].reduceSessions(3); c[3].reduceSessions(11); System.*out*.println("\nNew Team:"); **for** (int i = 0; i < c.length; i++) {  System.*out*.println(c[i]); } |  |

Old Team:

Coach: 123, sessions: 12, specialty: Jym

Coach: 234, sessions: 8, specialty: CrossFit

Athlete: 345, sessions: 12, Coach: 123, sessions: 12, specialty: Jym

Athlete: 456, sessions: 8, Coach: 234, sessions: 8, specialty: CrossFit

New Team:

Coach: 567, sessions: 8, specialty: Studio

Coach: 678, sessions: 8, specialty: CrossFit

Athlete: 345, sessions: 9, Coach: 123, sessions: 12, specialty: Jym

Athlete: 456, sessions: 0, Coach: 678, sessions: 8, specialty: CrossFit

C) Какие механизмы объектно-ориентированного программирования были использованы в этом вопросе? Укажите >= 3, объясните, где именно в коде они используются.  
   
  
**1. Наследование** (**Inheritance**)

* **Объяснение:** Наследование — это механизм, который позволяет одному классу (дочернему) перенимать свойства и методы другого класса (родительского). Это способствует повторному использованию кода и построению иерархии классов.
* **Где в коде:**  public class Athlete extends Coach

**2. Полиморфизм** (**Polymorphism**)

* **Объяснение:** Полиморфизм (в данном контексте) позволяет объектам дочерних классов использоваться там, где ожидаются объекты родительского класса, и при этом вызывать свою, специфическую реализацию методов.
* **Где в коде:**

1. **Массив родительского типа:** В классе TestEx2 создается массив типа Coach, который хранит как объекты Coach, так и объекты Athlete.

**Строка 4:** Coach[] c = new Coach[4];

**Строки 8-9:** c[2] = new Athlete(...) и c[3] = new Athlete(...). Здесь объект Athlete неявно приводится к типу Coach.

1. **Переопределение методов** (**Method Overriding**)**:** Оба класса имеют метод toString(). При вызове System.out.println(c[i]) для каждого элемента массива вызывается та версия метода toString(), которая соответствует фактическому типу объекта (либо Coach, либо Athlete).

**3. Композиция**(**Composition**)

**Объяснение:** Композиция — это создание сложных объектов путем включения в них объектов других классов. Это отношение типа "has-a" (у объекта "есть" другой объект).

**Где в коде:** Класс Athlete содержит в себе объект класса Coach в качестве одного из своих полей. **Строка 4:** **private** Coach personalCoach;

## B3. Media Приведения типов. Сравнения.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

|  |  |
| --- | --- |
| public class Media {  protected String title;  public Media(String t) {  this.title = t;  }  public String getTitle() {  return this.title;  }  public boolean matches(Media other) {  System.out.println("class Media");  return this == other;  }} | public interface Watchable {  boolean matches(Watchable other); } |
|  |
| public class Documentary extends Media implements Watchable {  **private** String topic;  public Documentary(String t, String topic) {  super(t);  this.topic = topic;  }  public boolean matches(Watchable other) {  System.out.println("class Documentary");  return (other instanceof Documentary) &&  ((this.title.equals(((Documentary)other).title)) ||  (this.topic.equals(((Documentary)other).topic)));  }} | |
| public class Movie extends Media implements Watchable {  **private** int duration;  public Movie(String t, int d) {  super(t);  this.duration = d;  }  public int getDuration() {return this.duration;}  public boolean matches(Movie other) {  System.out.println("class Movie");  return (other != null && this.title.equals(other.title) &&  this.duration == other.duration);  }  public boolean matches(Watchable other) {  System.out.println("class Movie");  return (other instanceof Movie) && matches((Movie)other);  } | |

Дано 4 действия, исходя из них найди вывод и поясни ошибки.

|  |  |
| --- | --- |
| Movie m1 = new Movie("Frozen", 95); |  |
| Object m2 = new Movie("Smurfs", 110); |  |
| Documentary d1 = new Documentary("Our ", "Nature"); |  |
| Media d2 = new Documentary("Ama", "Nature"); |  |

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Везде вначале идет System.*out*.println

|  |  |  |
| --- | --- | --- |
| ✅1. ((Watchable)d2).matches((Watchable)d1) | class Documentary, true | |
| Вызывается метод matches(Watchable) для объекта d2 (который на самом деле Documentary). Он сравнивается с d1 (тоже Documentary). |  | |
| ✅2.m1.matches((Movie)m2) | class Movie, false | |
| Вызывается метод matches(Movie) для объекта m1. Он сравнивается с m2 после явного приведения (המרה מפורשת) к Movie. |  | |
| ❌3. ((Movie)d2).getDuration() | RunTime Error | |
| Нельзя привести объект Documentary (фактический тип d2) к типу Movie, так как это "сестринские" классы, а не родитель-потомок. | | |
| ❌4. m2.getTitle()); | Compilation Error | |
| Переменная m2 имеет тип Object. У класса Object нет метода getTitle()  ✅ ((Movie)m2).getTitle() – так будет норм | | |
| ❌5. ((Documentary)m2).matches(d1) | Compilation Error  Тут сложно. | |
| Не смотря на то, что привести m2 к документари нельзя (Runtime), оно сначала спотыкается об вызов нескольких команд matches .  У класса есть несколько методов matches: Movie  matches(Media other) - унаследован от Media  matches(Movie other) - собственный метод  matches(Watchable other) - реализация интерфейса  Когда вы вызываете matches() с определенным аргументом, компилятор не может однозначно определить, какой метод использовать, поэтому выдает ошибку компиляции до того, как код будет выполнен. | | |
| ❌6.m1.title | Compilation Error | |
| поле protected, без него нет доступа. Чинится нахождением в 1package и без гетера. | | |
| ❌7.m1.matches(d1) | Compilation Error | |
| В методе matches нужно передать приведенный к интерфейсу объект.  ✅m1.matches((Watchable) d1) | | |
| ✅8. m1.matches((Watchable)m2) | class Movie, class Movie, false | |
| ❌9.m1.matches(m2) | Compilation Error | |
| нет метода matches(Object) в классе Movie  ✅m1.matches((Movie)m2) | | |
| ✅10. d1.matches((Watchable)d2) | Documentary, true | |
| ✅11. d1.matches(d2)  Есть унаследованный метод от родителя: matches(Media other) | class Media, false | |
| ❌12.(Documentary)d2).matches(m1) | Compilation Error | |
| m1 можно передать в оба метода:   * В matches(Media other) - потому что Movie extends Media * В matches(Watchable other) - потому что Movie implements Watchable   Компилятор не может определить, какой метод использовать!  ✅ ((Documentary)d2).matches((Media) m1)  или ((Documentary)d2).matches((Watchable) m1) | | |
| ✅13. ((Media)m1).getTitle() | | Frozen |
| ✅14. ((Movie)m2).matches((Media)m2) | | class Media, true |

## M2025\_KA

## S4. Cars – приведения типов.

|  |  |
| --- | --- |
| Vehicle v1 = new Vehicle();  Vehicle v2 = new Car();  Vehicle v3 = new Truck();  Car c = new Truck (); |  |

A(

1. Car c1 = new Vehicle (); -- compilation. חידור

Car יורש מ Vehicle ןלא להיפך לכן לא ניתן ליצור נק; מבט של Car (בן) לאוביקט מסוג Vehicle (בן)

2. Car c2 = v1; -- compilation. חידור.. то же что сверху

3. Car c3 = (Car) v2; ++ удачный даункастинг.

4. Car c4 = (Car) v3; ++ удачный ап и потом даун кастинг.

5. Car c5 = (Car) v1; -- RunTime. שגיאה זמן ריצה. המרה מטה מפורשת, אבל לא לצורך שחזור

6. Motorcycle m = new Motorcycle();

Vehicle v6 = m;

Car c6 = (Car) v6; -- RunTime. שגיאה זמן ריצה. המרה מטה מפורשת, אבל לא לצורך שחזור

7. Vehicle v7 = (Car)(new Truck ()); ++ удачный двойной upCasting

8. Car c8 = (Vehicle)(new Truck ()); -- . המרה מטה לא מפורשת compilation. חידור

B)

Где нужно добавить параметр double speed() чтобы можно было вычислить среднюю скорость у Motorcycle & RacingCar?

Vehicle **abstract** -> Public **abstract** double speed(); ->

Motorcycle & RacingCar Override public double speed()

C)

Найти самый быстрый мотоцикл в массиве и вернуть его

public static Motorcycle maxSpeed(Vehicle[] arr) {

Motorcycle maxM = null;

double maxSpeed = -1;

**for**(int i=0; i<arr.**length**; i++) {

**if**(arr[i] instanceof Motorcycle) {

**if**(arr[i].speed() > maxSpeed) {

maxSpeed = arr[i].speed();

maxM = arr[i];

}

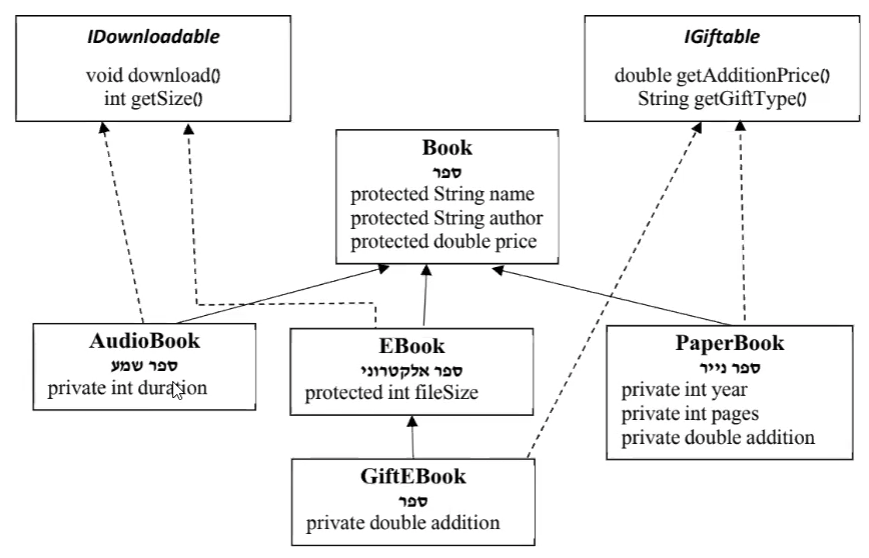
}

}

return maxM;

}

## Ebook. Интерфейсы



public **abstract** class Book {

}

public class AudioBook extends Book implements IDownloadable {

public void download() { ... }

public int getSize() { ... }

}

public class EBook extends Book implements IDownloadable {

public void download() { ... }

public int getSize() { ... }

}

public class GiftEBook extends EBook implements IGiftable {

public double getAddOn() { ... }

public String getGiftType() { ... }

}

**B**) **есть 3 константы. Куда их разместить?**

 final double EBOOK\_FACTOR = 0.8;

final double AUDIOBOOK\_FACTOR = 1.2;

final double PAPERBOOK\_FACTOR = 1.0;

Book - הוא מחלקה מפורשת משותפט לכל המחלקות ולכן הייתי מגדירה את הקבועים בתוכה וקשר ללוגיקה של מחיר ספר שמוגדת בתכונה ב Book

Book - это абстрактный класс, общий для всех классов, и поэтому я бы определила константы внутри него, связанные с логикой цены книги, которая определена в Book."

public **abstract** class Book { *//* это не надо писать в ответе. Чисто для себя.

*// Константы для расчета цены*

    protected static final double EBOOK\_FACTOR = 0.8;

    protected static final double AUDIOBOOK\_FACTOR = 1.2;

    protected static final double PAPERBOOK\_FACTOR = 1.0;

*// Поля класса*

    protected String name;

    protected String author;

    protected double price;

}

С) **Написать метод**(**ы**) **для расчета цены книги с учетом коэффициентов:**

Book:

Public **abstract** double getPrice();

AudioBook, Ebook, PaperBook:

@Override

Public double getPrice(){

Return super.price \* \_\_\_\_\_; \_\_\_ - тут СТАТИК из пункта выше

*// В классе EBook*

public double getCalculatedPrice() {

    return price \* EBOOK\_FACTOR;

}

**D**) **Напишите внешнюю функцию, которая получает массив книг и печатает подходящие под бюджет клиента.**

*//* in – arr books, double budget

*//* out – print books details

public static void printBooksBudget(Book[] arr, double budget) {

int count = 0;

double totalSum = 0;

**for** (int i=0; i<arr.**length**; i++) {

**if** (arr[i].getPrice() <= budget) {

syso(arr[i]);

} } }

**E**) **"Напишите внешнюю функцию, которая получает массив книг, описывающий инвентарь магазина. Функция выведет количество книг, которые можно купить в подарочной упаковке."**

public static void printGiftableBooks(Book[] inventory) {

int giftableCount = 0;

**for** (Book book : inventory) {

**if** (book instanceof IGiftable) {

giftableCount++;

}

}

System.out.println("Количество книг, доступных в подарочной упаковке: " + giftableCount);

}

## S10\_Подъебка. AA AB

|  |  |
| --- | --- |
| public class AA {  public AA(){  System.out.println("AA constructor");  }  public void one(){  System.out.println("one in AA");  }  public void two(){  one();  } } | public class BB extends AA{  public BB(){  System.out.println("BB constructor");  }  public void one(){  System.out.println("one in BB");  }  public void two(){  System.out.println("two in BB");  }  public void three(){  super.two();  }} |

А) Что выведет?

|  |  |  |
| --- | --- | --- |
| AA a = new AA();  BB b = new BB();  AA ab = new BB();  a.one();  ab.one();  b.two();  a.two();  ab.two();  b.two();  ((BB)(ab)).three(); | *//* 1. AA constructor  *//* 2. AA constructor -> BB constructor  *//* 3. AA constructor -> BB constructor  *//* 4. one in AA *//* 5. one in BB  *//* 6. two in BB  *//* 7. one in AA  *//* 8. two in BB  *//* 9. two in BB  *//* 10. one in BB -> тут ПОДЬЕБКА  *//* 11. one in BB -> тут такая же ПОДЬЕБКА |  |

((BB)(ab)).three(): 1. Успешный downCasting

2. объект BB зовет super.two().

3. в АА метод two() зовет метод one() – НО НЕ ИЗ АА, а из BB. Это Override

Выодит в итоге "one in BB"

B) a.three(); - ошибка компиляции

Three לא מוגדרת ב מבט AA

((BB)a).three(); - ошибка чтенияשגיאה זמן ריצה. המרה מטה מפורשת אך לא לצורך שחזור.

С) **Напишите функцию, которая получает массив ссылок на объекты типа** Object**. Функция должна напечатать сколько объектов являются типом BB, сколько объектов являются типом AA и не типом BB, и сколько объектов не являются типом AA.**

public static void countObjectTypes(Object[] arr) {

int countBB = 0; int countAA\_NotBB = 0; int countNotAA = 0;

**for** (Object **obj** : arr) {

**if** (**obj** instanceof BB) { countBB++; }

**if** (**obj** instanceof AA && !(**obj** instanceof BB)) { countAA\_NotBB++; }

**if** (!(**obj** instanceof AA)) { countNotAA++; }

}

System.out.println(countBB + “ ”+ countAA\_NotBB+” “ + countNotAA);

}

# M2021\_A

## S1\_Tablet Store – simple array oop

Класс Tablet, представляющий планшетный компьютер. Атрибуты (свойства) класса:  
name —тип String. kind — модель, тип String (строка).

system — ос, тип char (символ). ('W' - Windows, 'A' - Android, 'I' — iOS).  
size — double. price — double  
Можно предположить, что в классе уже определены:  
Конструктор, принимающий параметры для всех атрибутов. Методы get/set для всех атрибутов.Метод toString. Нет необходимости реализовывать эти методы.

|  |
| --- |
| **А. Напишите метод** publicboolean **isSame**(**Tablet other**): Метод возвращает true, если имя (name), модель (kind) и размер (size) объекта, для которого был вызван метод, идентичны имени, модели и размеру объекта other. В противном случае метод должен вернуть false. |
| public boolean isSame(Tablet other){  **if** (other.getName().**equals**(this.getName())  && other.getSize() == (this.getSize())  && other.getKind().**equals**(this.getKind())  ) return true;  return false; } |

Класс Store

|  |  |
| --- | --- |
| Класс Store представляет собой склад (или магазин). Атрибуты (поля) класса: tablets — массив объектов типа Tablet. Размер этого массива — 1000. systems — массив целых чисел. В каждой ячейке этого массива хранится к-во планшетов, сгруппированных по операционной системе. Порядок: с Windows. с Android. с iOS. | Фрагмент кода класса Store:  public class Store { **private** Tablet[] tablets; **private** int[] systems;  public Store() {  tablets = new Tablet[1000];  systems = new int[3];  }  *//* ... (дальнейшая реализация) } |

**Б. Напишите метод**publicboolean **addTablet**(**Tablet tab**)Метод принимает в качестве параметра планшет tab и добавляет его на склад (Store). Логика работы метода следующая:

* **Если на складе нет места** для добавления нового товара, метод должен вернуть false.
* **Если на складе уже существует товар** с таким же именем, моделью и размером (как у tab), метод должен **обновить цену** этого существующего товара, установив **наибольшую** из двух цен (цены существующего товара и цены tab). После этого метод должен вернуть true.
* **Если товар, аналогичный tab, на складе не существует**, метод должен **добавить** его в массив товаров, обновить соответствующую статистику на складе (подразумеваются счетчики) и вернуть true.

**Важное замечание:** Объекты в массиве должны храниться **последовательно, без "дыр"** (то есть без ячеек со значением null между реальными объектами).

1. добавляем в реализацию класса Store **private** int tabletCount; и в конструктор tabletCount=0;

|  |
| --- |
| public boolean addTablet(Tablet tab){  *//* --- ШАГ 1: Поиск существующего товара ---  *//* Проходим только по тем планшетам, которые уже есть на складе (до tabletCount)  **for** (int i=0; i<this.tabletCount; i++){  **if** (this.tablets[i].isSame(tab)) {  *//* Товар найден. Обновляем цену на максимальную из двух.  double newPrice = Math.*max*(this.tablets[i].getPrice(), tab.getPrice());  this.tablets[i].setPrice(newPrice);  return true;  }  } |
| *//* Если мы дошли до сюда, значит, такого товара на складе нет.  *//* --- ШАГ 2: Проверка, есть ли место на складе ---  **if** (this.tabletCount >= this.tablets.**length**) {  *//* Склад полон (счетчик достиг размера массива).  return false;  } |
| *//* --- ШАG 3: Добавление нового товара ---  *//* Место есть, и товар уникален. Добавляем его в первую свободную ячейку.  this.tablets[this.tabletCount]=tab;  this.tabletCount++;  return true; } |

**Г. Напишите метод**publicint **sortStore**()Метод **сортирует массив товаров** (tablets) таким образом, чтобы в его начале располагались планшеты с операционной системой **Windows**, за ними — устройства с операционной системой **Android**, и в самом конце — устройства с операционной системой **iOS**. Метод должен **вернуть количество свободных мест** на складе.

|  |
| --- |
| public int sortStore(){ *//* Если на складе нет товаров (или всего один), он уже "отсортирован".  **if** (this.tabletCount <= 1) return this.tablets.**length** - this.tabletCount;   *//* 1. Создаем временный массив размером с реальное количество планшетов.  Tablet[] sortedTablets = new Tablet[this.tabletCount];  int currentIndex = 0; *//* "Указатель" на текущую свободную ячейку в sortedTablets.  *//* 2. Первый проход: ищем и копируем все планшеты с WINDOWS.  **for** (int i = 0; i < this.tabletCount; i++) {  **if** (this.tablets[i].getSystem() == OperatingSystem.*WINDOWS*) {  sortedTablets[currentIndex] = this.tablets[i];  currentIndex++;  }  } и дальше такие же циклы для остальных ос.  *//* 5. КОПИРУЕМ ДАННЫЕ ОБРАТНО и копируем его на то же место в основной массив.  **for** (int i = 0; i < this.tabletCount; i++) {  this.tablets[i] = sortedTablets[i];  }  *//* 6. Возвращаем количество свободных мест.  return this.tablets.**length** - this.tabletCount; } |

## S3\_DressCompany – simple arr erusha

Есть родительский класс Clothes(у него наследники: dress и т.д.). Есть отдельный класс Book. Подъебка в том, что у Book и Clothes есть getPrice() и свойство double price. Надо все товары забить в один массив и посчитать итоговую сумму.

public double sum(){  
 double sum=0;  
 **if** (arr.**length**<1) return 0;  
 **for** (Object item : arr){  
 **if** (item instanceof Book) sum+= ((Book)item).getPrice();  
 **else** **if** (item instanceof Clothes) sum+= ((Clothes)item).getPrice();  
 }  
 return sum;  
}

## S4\_A-B – «если-бы да-кабы»

|  |  |
| --- | --- |
| public class A { } | public class B extends A{  public B(){  System.out.println("B constructor");  } } |

А)  
\*\*1. Конструктор B вызовет пустой конструктор A\*\* \*\*ВЕРНО\*\* ✓  
Поскольку в конструкторе B нет явного вызова `super()`, Java автоматически вставляет `super()` в качестве первой строки. Класс A не имеет явно определенного конструктора, поэтому Java создает конструктор по умолчанию без параметров.  
\*\*2. Пустой конструктор Object не будет вызван, потому что у A нет явного вызова этого конструктора\*\* \*\*НЕВЕРНО\*\* ✗  
Конструктор Object будет вызван автоматически. Даже если в классе A нет явного вызова `super()`, Java автоматически вставляет этот вызов в конструктор по умолчанию класса A. Каждый класс в Java неявно наследуется от Object.  
\*\*3. Пустой конструктор Object будет вызван после печати строки "B constructor"\*\* \*\*НЕВЕРНО\*\* ✗  
Конструктор Object будет вызван ДО печати строки "B constructor". Конструкторы родительских классов всегда выполняются перед кодом дочернего конструктора.  
\*\*4. Пустой конструктор B вызовет пустой конструктор A, а затем будет напечатана строка\*\* \*\*ВЕРНО\*\* ✓

Б)

|  |  |  |
| --- | --- | --- |
| public class A {  \*\*\*\*\* int value; } | public class B extends A{  public B(){  System.out.println("B constructor");  } } | public class C extends B{ } |

\*\*1. Из класса C нет доступа к свойству value класса A, даже если value определено как protected, поскольку речь идет о двух уровнях наследования\*\* \*\*НЕВЕРНО\*\* ✗  
Модификатор доступа `protected` позволяет доступ из любого класса-наследника, независимо от количества уровней наследования. Класс C наследуется от B, который наследуется от A, поэтому C имеет доступ к protected-членам A.  
\*\*2. Из класса C есть доступ к свойству value только при условии, что свойство value в классе A определено как protected\*\* \*\*НЕВЕРНО\*\* ✗  
Это утверждение неполное. Доступ к свойству value из класса C возможен при следующих условиях:  
- `public` - доступ есть  
- `protected` - доступ есть (так как C наследуется от A)  
- `package-**private**` (по умолчанию) - доступ есть, если все классы в одном пакете  
- `**private**` - доступа нет  
\*\*3. Из класса C есть доступ к свойству value класса A с помощью команды super.super.value\*\* \*\*НЕВЕРНО\*\* ✗  
В Java не существует конструкции `super.super`. Ключевое слово `super` может использоваться только для обращения к непосредственному родительскому классу. Для доступа к свойству value из класса C нужно использовать просто `value` (если доступ разрешен) или `super.value` (обращение к родительскому классу B).  
\*\*4. Из класса C есть доступ к свойству value, поскольку класс C косвенно наследуется от класса A, независимо от разрешения доступа к свойству\*\* \*\*НЕВЕРНО\*\* ✗  
Доступ к свойству всегда зависит от модификатора доступа. Если свойство value в классе A определено как `**private**`, то из класса C к нему доступа не будет, даже при наследовании. Модификаторы доступа определяют видимость членов класса.  
\*\*Правильный способ доступа к value из класса C:\*\*  
- Если `value` имеет модификатор `public`, `protected` или package-**private** (и классы в одном пакете), то из C можно обратиться просто: `value`  
- Если нужно явно указать, что обращение идет к унаследованному свойству: `this.value`

В) ((A)b).myFun() **Анализ утверждений:**

**1. Операция myFun определена в классе A** **ВЕРНО** ✓

Поскольку код ((A)b).myFun() успешно компилируется, это означает, что после приведения типа к A компилятор находит метод myFun() в классе A. Если бы метод не был определен в A, компиляция бы завершилась ошибкой.

**2. B - это класс, наследующий класс A** **ВЕРНО** ✓

Из предоставленного кода видно public class B extends A, что подтверждает наследование B от A. Кроме того, приведение типа (A)b возможно только если B является подклассом A или реализует интерфейс A.

**3. A - это класс, наследующий класс B** **НЕВЕРНО** ✗

Это противоречит утверждению 2 и логике наследования. Из кода видно, что B наследуется от A (extends A), а не наоборот. В Java класс не может одновременно наследоваться от другого класса и быть его родителем.

**4. Операция myFun определена и в классе A, и в классе B** **ВОЗМОЖНО ВЕРНО** ⚠️

## S5\_A-B – hard ERUSHA.

|  |  |
| --- | --- |
| public class A {  public void f(){  System.out.println("A.f");  }  public void g(){  f();  } } | public class B extends A {  public void f(){  System.out.println("B.f"); }  public void g(){  System.out.println("B.g");  }  public void superG(){  super.g();  } } |

A a = new A();  
B b = new B();  
A ab = new B();  
a.f(); *//* A.f  
ab.f(); *//* B.f  
b.f(); *//* B.f  
a.g(); *//* A.f  
ab.g(); *//* B.g  
b.g(); *//* B.g  
((B)(ab)).superG(); *//* B.f - главная подьебка с override.  
b.superG(); *//* B.f - главная подьебка с override.

*//* a.superG(); - Compilation error -нет доступа к методам наследников.  
  
*//* ((B)a).superG(); ## Объяснение ошибки:  
\*\*1. Проблема приведения типа:\*\*  
- - это переменная типа `a``A`  
- `(B)a` - попытка привести объект типа к типу `B` `A`  
- Это \*\*недопустимое приведение\*\*, поскольку `B` является подклассом , а не наоборот `A`  
*//*\*\*2. Правила приведения типов в Java:\*\*  
*//*- ✅ \*\*Восходящее приведение (Upcasting)\*\*: `B` → (подкласс к родительскому классу) - всегда безопасно `A`  
*//*- ❌ \*\*Нисходящее приведение (Downcasting)\*\*: → `B` (родительский класс к подклассу) - возможно только если объект реально является экземпляром `A`

## S6\_Banana – hard EQUALS.

|  |  |
| --- | --- |
| public class Apple {  **private** int weight;  public Apple(int w){  weight=w;  }   public boolean **equals**(Apple other) {  return ((other!=null) && weight==other.weight);  } } | public class Banana {  **private** int weight;  public Banana (int w) {  weight = w;  }  public int getWeight () {  return weight;  }  public boolean **equals** (Object other) {  return ((other != null) &&  (other instanceof Banana) &&  (weight == ((Banana)other).weight));  }} |

А)*//* - \*\*Banana\*\*: Переопределение (Override) ✓

**equals**(Object other) является **переопределением** (**overriding**) метода **equals** из класса Object

* Сигнатура метода точно соответствует методу из родительского класса Object

*//*- \*\*Apple\*\*: Перегрузка (Overload) ✓

* Это НЕ переопределение метода **equals** из класса Object, поскольку параметр имеет тип , а не Object Apple
* Класс фактически имеет ДВА метода **equals**:
  1. Унаследованный **equals**(Object) от класса Object
  2. Собственный **equals**(Apple) - это перегрузка

|  |  |
| --- | --- |
| Apple a1 = new Apple (10);  Object a2 = new Apple (10);  Banana b1 = new Banana (10);  Object b2 = new Banana (10); |  |
| 1. syso(a1.weight) | ошибка compilation. **private**. |
| 2. syso (((Banana)a2).getWeight()) | \*\*runtime error\*\* `ClassCastException` |
| 3. a1.**equals**(a2) | False; - метод сравнивает епл и епл, а не obj  *//* Компилятор ищет метод `**equals**(Object)` В классе такого метода нет! `Apple` Вызывается унаследованный Object.**equals**(Object) |
| 4. a1.**equals**((Apple)a2)) | True; |
| 5. a2.**equals**(a1) | False; Компилятор видит: Object.equals(Object) |
| 6. b1.equals(b2) | True; метод сравнивает банан и обжект  Опция прописана в ‘B’ классе. |
| 7. b2.**equals**(b1) | True - метод сравнивает банан и обжект  `b2` имеет тип `Object`, но содержит `Banana` *//* - Вызывается переопределенный метод `equals` из класса `Banana` |
| 8. a1.**equals**((Banana)b2) | False. не ошибка. оно идет в метод родителя  Object.**equals**(Object)  public boolean **equals**(Object **obj**) {  return (this == **obj**); } *//* Сравнение ссылок |
| 9. b1.**equals**((Banana)a2); | \*\*runtime error\*\* `ClassCastException` |
| 10. b1.equals((Apple)a2)) | False; |

# EKZAMEN

# Буквы, erusha, если бы да кабы.

В этом вопросе 4 пункта. Между нет никакой связи. Ответьте на вопросы.

Изображение выглядит как символ, линия, зарисовка, оригами

Содержимое, созданное искусственным интеллектом, может быть неверным.

1.А) Дано классы X и Y. Y наследует от X. Знаем, что следующая строка компилируется верно:

**X x1 = new Y**()**;**

Напротив каждого утверждения оцените правильно или ошибка. Обьясните ваши ответы. Если ошибка – почините его, если возможно.

|  |  |
| --- | --- |
| i) Object o = x1;  + | (i -לא ניתן לכתוב את ההוראה, מכיוון שדרושה המרה מפורשת.  Невозможно выполнить инструкцию, так как требуется явное преобразование (типов) |
| Нет ошибки, потому что X наследует от Object. upCast (i - | |
| ii) X x2 = x1;  + | (ii - לא ניתן לכתוב את ההוראה, מכיוון שדרושה המרה מפורשת. |
| Нет ошибки, потому что параметр х1 это типа Х | |
| iii) Y y1 = x1; - | (iii + לא ניתן לכתוב את ההוראה, מכיוון שדרושה המרה מפורשת. |
| Y y1 = (Y) x1; + | אפשר לעשות רק עם המרה מפורשת בחזרה ל-Y |

1.B) Есть 4 строки, которые компилируются. У нас есть 5 вариантов с деревом наследования. Нужно нарисовать UML, если нельзя скомпилировать – обяьсните почему. Напишите вид ошибки.

|  |
| --- |
| (1) G g = new E(); |
| (2) F f1 = (F)(new E()); |
| (3) F f2 = new D(); |
| (4) E e = new D(); |

|  |  |  |
| --- | --- | --- |
| Изображение выглядит как диаграмма, Прямоугольник, дизайн  Содержимое, созданное искусственным интеллектом, может быть неверным. | I | |
| (1) G g = new E(); | שגיאת חידור (compilation). G יורשת מ .E |
| (2) F f1 = (F)(new E()); | שגיאת ריצה. F יורשת מ .E ומנסים המרה למטה. |
| (3) F f2 = new D(); | שגיאת חידור (compilation). F אין קשר ל .D |
|  | II | |
| Все работает | |
| Изображение выглядит как диаграмма, дизайн  Содержимое, созданное искусственным интеллектом, может быть неверным. | III | |
| (1) G g = new E(); | שגיאת חידור (compilation). G יורשת מ .E |
| (2) F f1 = (F)(new E()); | שגיאת ריצה. F יורשת מ .E ומנסים המרה למטה. |
| Изображение выглядит как снимок экрана, дизайн  Содержимое, созданное искусственным интеллектом, может быть неверным. | III | |
| Невозможное наследование | |
| Изображение выглядит как диаграмма, дизайн  Содержимое, созданное искусственным интеллектом, может быть неверным. | IV | |
| (1) G g = new E(); | שגיאת חידור (compilation). G יורשת מ .E |
| (2) F f1 = (F)(new E()); | שגיאת ריצה. F יורשת מ .E ומנסים המרה למטה. |
| Изображение выглядит как дизайн  Содержимое, созданное искусственным интеллектом, может быть неверным. | V | |
| (2) F f1 = (F)(new E()); | שגיאת ריצה. F יורשת מ .E ומנסים המרה למטה. |
| (3) F f2 = new D(); | שגיאת חידור (compilation). F אין קשר ל .D |

1.C) Выберите один из следующих вариантов и укажите, какой из них является наиболее подходящим в соответствии с принципами объектно-ориентированного программирования (ООП). Обоснуйте свой ответ.

|  |  |
| --- | --- |
| public class City{  \_\_\_(\*)\_\_\_ int employees = 20;  }  public class Company extends City{  public void poll(){  syso(”There are ” + employees”);  }} | 1) **private**, public или protected. 2) public или **private**. 3) public или protected. **4**)protected**.** 5) public. 6) Написать только **private**. |

מגדירים את התכונה רק כ- protected. כך שרק מחלקות שיורשים מ- City יהיה להן גישה ישירה לתכונה.

public тоже скомпилирует, но это нарушит не «принцип», а скорее пожелание инкапсуляции – где доступ должен быть минимально необходимым.

1.D) Исходя из определения класса, реализуйте в мейне команды, чтобы получить такой вывод:

*//* 1) val 3, count 5

*//* 2) val 2, count 5

*//* 3) val -999, count 5

*//* 4) val 5, count 5

|  |  |
| --- | --- |
| public class Stam {  **private** int val;  **private** static int *count* = 0;  public Stam() {  *count*++;  this.val=*count*;  }  public Stam(boolean over){  **if** (!over){  *count*++;  this.val = *count*;  }**else**{  this.val = -999;  } }  public void display(){  System.out.println("val=" + val+", count ="+*count*);  }{ | Stam s1 = new Stam(); Stam s2 = new Stam(); Stam s3 = new Stam(); Stam s4 = new Stam(); Stam s5= new Stam();  *//* 1) val 3, count 5 s3.display();  *//* 2) val 2, count 5 s2.display();   Stam d5 = new Stam(true);  *//* 3) val -999, count 5 d5.display(); *//* 4) val 5, count 5 s5.display(); |

הבנאי הראשון תמיד סופר ומוסיף כמה אוביאקטים אותחלו מסוג Stam. ושומר את המספר הזו, בערך של val.

בבנאי השני שמקבל ערך בוליאני, אם הארך TRUE, אפשר לצור את האובריאקט בלי שינוי לסופר, אבל ערךval 999 -

Inteface.



**А.** Для каждого из пяти представленных ниже классов необходимо определить:

**a.** Заголовок (сигнатуру) класса

**b.** Свойства (атрибуты/поля) класса

**c.** Заголовки (сигнатуры) всех методов, которые должны быть в классе.(**15 баллов**)

|  |  |
| --- | --- |
| public interface Colorable {  void colortofill(String color); } | public interface Rotatable {  void toleft(double degree);  void toright(double degree); } |
| public class Figure2D {  protected String color;  protected Point coord1; } | public class Circle extends Figure2D implements Colorable{  **private** int radius;  public void colortofill(String color) {} } |
| public class Square extends Figure2D implements Colorable, Rotatable{  protected int radius;  public void colortofill(String color) {};  public void toleft(double degree) {}  public void toright(double degree) {} } | public class Line extends Figure2D implements Rotatable{  **private** Point coord2;  public void toleft(double degree) {}  public void toright(double degree) {} } |
| public class Point {  **private** int x;  **private** int y; } |  |

**Б.**  Для каждой из строк (1–15) необходимо указать:

* Является ли строка корректной или нет.
* Если строка некорректна — необходимо обосновать причину и, если возможно, предложить исправление.
* Указать тип ошибки: ошибка компиляции / ошибка времени выполнения



|  |  |
| --- | --- |
| 1. Figure2D c = new Circle("red", new Point(50,50), 8); | + |
| 2. Line line = new Line("blue", new Point(100,100), new Point(200,200)); | + |
| 3. Square d2 = new Square("orange", new Point(45,45), 4); | + |
| 4. Colorable col1 = new Colorable();   * Нельзя создать объект из интерфейса. | compilation error |
| 5. Rotatable rot1; | + |
| 6. Colorable col2 = d2; - пример המרה מרומזת | + |
| 6.1. col2.toleft(5); - не смотря на то, что d2 – квадрат, компилятор  Видит в нем только объект интерфейса Colorable. | compilation error |
| 7. rot1 = line; | + |
| 8. col2 = (Colorable)c; | + |
| 9. rot1.setColor("green");   * Нет такого метода для него. Можно rot1.toleft() | - compilation error |
|  |
| 10. col2.colortofill("red"); | + |
| 11. col2.toleft(Math.PI/2);  сol2 после 8й строки – это ТОЛЬКО объект интерфейса Colorable | - compilation error |
|  |
| 12. ((Square)d2).toleft(Math.PI/4); | + |
| 13. Figure2D d = c; | + |
| 14. ((Line)d).setColor("red");  המרה מפורשת למטה לא לצורך שיחזור.  ((circle)d).setColor(“red”) – а так можно. | - runtime error |
| 15. d.colortofill("Yellow");  Вы пытаетесь вызвать метод colortofill() у переменной d.  Компилятор смотрит на **тип переменной d**, а он — Figure2D.  Компилятор проверяет: "Есть ли в классе Figure2D метод colortofill()?   * **Ответ: Нет.** Метод colortofill() определен в интерфейсе Colorable, а не в родительском классе Figure2D.   ((Colorable)d).colortofill("Yellow") – а так можно. | - compilation error |



**В.** Напишите внешнюю функцию с именем handleFigures, которая получает массив объектов Figure2D и выполняет следующие действия:

* Печатает точку cood1 всех фигур синего цвета ("blue").
* Если фигуру можно залить, закрашивает ее в красный цвет ("red").
* Если фигуру можно вращать, поворачивает ее на 30 градусов влево.
* Если фигура является Circle (Кругом), печатает радиус круга.
* Если это Line (Линия), печатает ее вторую точку.

Функция возвращает количество фигур, которые можно вращать, но нельзя залить.

public static int handleFigures(Figure2D[] arr) {  
 Figure2D curr;  
 int count = 0;  
 **for**(int i=0; i< arr.**length**; i++) {  
 curr = arr[i];  
 *//* 1: Напечатать координаты синих фигур  
 **if** ("blue".**equals**(curr.getColor())) System.out.print(curr.getPoint());

*//* либо  
**if** (curr.getColor().**compareTo**("blue")==0) System.out.print(curr.getPoint());

*//* 2: Если форму можно заполнить, заполнить цветом "red"  
 **if** (curr instanceof Colorable) {  
 Colorable c = (Colorable) curr;  
 c.colortofill("red");  
 }  
 *//* 3: Повернуть фигуру на 30 градусов влево, если возможно  
 **if** (curr instanceof Rotatable) {  
 Rotatable r = (Rotatable) curr;  
 r.toleft(30);  
 }  
 *//* 4: Если это Circle, напечатать радиус  
 **if** (curr instanceof Circle) System.out.println( ((Circle)curr).getRadius() );  
 *//* 5: Если это Line, напечатать вторую точку  
 **if** (curr instanceof Line) System.out.println(((Line) curr).getCoord2());  
 *//* 6: Подсчет фигур, которые можно повернуть, но нельзя закрасить  
 **if** (curr instanceof Rotatable && !(curr instanceof Colorable)) {  
 count++;  
 }  
 }  
 return count;  
}

# A B. Маакав

Есть 2 класса А и B

|  |  |
| --- | --- |
| public class A {  **private** int val;  public A(){  this.val=1;  }  public A(int val){  this.val = val;  }  public int getVal(){  return this.val;  } | public class B extends A {  **private** String st; public B() {  this.st = "B"; } public B(String st, int val){  super(val);  this.st=st;  } public String getSt(){  return this.st; } |
| public boolean **equals**(Object other){  System.out.println("A-Object");  **if** (other instanceof A)  return this.val == ((A)other).val;  return false;  } } |
| public boolean **equals**(Object other) {  System.out.println("B-object");  **if** (other instanceof B){  return this.st.**equals**( ((B)other).st) && this.getVal() == (((B) other).getVal());  }  return false;  } |
| public boolean **equals**(A other){  System.out.println("B-A");  **if** (other instanceof B)  return this.st.**equals**( ((B)other).st) && this.getVal()==((B)other).getVal();  return false;  }} |
| public boolean **equals**(B other){  System.out.println("B-B");  return this.st.**equals**(other.st) && this.getVal()==other.getVal();  }} |

A) нарисовать маакав на строки:

|  |  |
| --- | --- |
| A a1 = new A(); | Изображение выглядит как Шрифт, белый, текст, символ  Содержимое, созданное искусственным интеллектом, может быть неверным. |
| A a2 = new A(3); | Изображение выглядит как Шрифт, белый, Графика, символ  Содержимое, созданное искусственным интеллектом, может быть неверным. |
| A ab = new B(); | Изображение выглядит как Шрифт, текст, диаграмма, белый  Содержимое, созданное искусственным интеллектом, может быть неверным. |
| B b1 = new B("B", 1); | Изображение выглядит как Шрифт, диаграмма, текст, число  Содержимое, созданное искусственным интеллектом, может быть неверным. |
| B b2 = new B("B", 3); | Изображение выглядит как текст, Шрифт, число, снимок экрана  Содержимое, созданное искусственным интеллектом, может быть неверным. |

Б) что напечатает в строках и обьясните:

1) **if** (a1.**equals**(b1)) System.out.println(11); *//* **"A-**Object**", 11**

A-Object – למחלקה A קיים דריסה, לפעולת 11 **equals** של מחלקה Object, לכן הולכים לשם.

2) **if** (b1.**equals**(a1)) System.out.println(12); *//* **"B-A".**

יש פעולה **equals** שמקבלת A. אבל היא מחזירה false המקרנ הזה.

Т.к. а1 – не экземпляр B. false на вывод 12.

3) **if** (a1.**equals**(ab)) System.out.println(13); *//* **"A-**Object**", 13**

יש למחלקה A פעולת **equals** שמרבלת Object אבל הפניה שנשלחו היא בפעול instanceof A לכן הוחזר true

4) **if** (ab.**equals**(a1)) System.out.println(14); *//* **"B-object"**

**ПОДЪЕБКА!! – ab – это overloading объект типа B. И на него в JAVA по умолчанию вызывается equals**(Object **obj**)**. Внутри сраного B – 3 метода, и когда у нас подьебаный ab – то** Object **obj, а не как в** стандартном2) случае

5) **if** (b1.**equals**(ab)) System.out.println(15); *//* **"B-A", 15**

**ПОДЪЕБКА!! – ab – здесь это ПЕРЕМЕННАЯ типа А.** A ab = new B();

6) **if** (ab.**equals**(b1)) System.out.println(16); *//* “B-object”, 16

**ПОДЪЕБКА!! – ab – это overloading объект типа B.**

7) **if** (a1.**equals**(a2)) System.out.println(17); *//* A-Object

8) **if** (b1.**equals**(b2)) System.out.println(18); *//* B-B  
  
В) Напишите функцию matchCount которая получает массив типа Object[]. ф-я считает и возвращает сколько из объектов в массиве это экземпляры класса “B” и сколько из них отвечают двум требованиям: val >2 и st!=”B”.

public static int matchCount(Object[] arr){  
 int count = 0;  
 **if** (arr == null || arr.**length** == 0) return 0;   
 **for** (int i=0; i<arr.**length**; i++){  
 Object **obj** = arr[i];  
 **if** (**obj** instanceof B){  
 B b\_instance = (B) **obj**;  
 **if** (b\_instance.getVal()>2 && ( !"B".**equals**(b\_instance.getSt()))) count++;  
 } } return count;}

**Г**) Используется ли принцип **Перегрузки** (**Overloading**) **העמסה** - в классах A и B? **Ответ:Да, используется.**

1. **В конструкторах** (**в обоих классах**)**:**

* В классе A есть два конструктора: A() и A(int val). Это перегрузка конструктора.
* В классе B есть два конструктора: B() и B(String st, int val). Это тоже перегрузка конструктора.

1. **В методе equals**(**в классе B**)**:**

* В классе B вы написали **три разных метода** с именем equals, но с разными параметрами:
* equals(Object other)
* equals(A other)
* equals(B other)
* Это классический пример **перегрузки метода**.

**Д**)Используется ли принцип **Переопределения** (**Overriding**) **דריסה**  в классах A и B? **Ответ:Да, используется.**

1. **Класс A переопределяет метод из Object:**

* В классе A есть метод public boolean equals(Object other).
* Этот метод имеет точно такую же сигнатуру (имя и параметры), как и метод equals в самом главном классе Object.
* Следовательно, класс A **переопределяет** стандартное поведение метода equals.

1. **Класс B переопределяет метод из A:**

* В классе B тоже есть метод public boolean equals(Object other).
* Поскольку B наследуется от A (extends A), этот метод **переопределяет** метод equals(Object other) из класса A. Класс B предоставляет свою, более специфичную логику сравнения.