НАСЛЕДЯВАНЕ

ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

- Въведение
- Примери
- Наследяване на методи
- Наследяване на данни
- Полиморфизъм
- Контрол на достъп: обобщение
- Йерархии от класове
- Абстрактни класове
- Ключови думи: Extends, Protected, Super, Final

ВЪВЕДЕНИЕ

- Многократно използване
  - Много съществена идея, която предоставя повече от копиране и промяна на кода
- В Java (както всичко друго) решението е свързано с класове
  - Използваме повторно код за създаване на нови класове
  - Без да ги създаваме от самото начало, а използваме съществуващи класове
- Принципно два подхода:
  - Композиция
    - В новия клас използваме обекти от вече съществуващи класове
    - Използва се повторно функционалността на кода
  - Наследяване
    - Новият клас се създава като вариант на съществуващ клас
      - Използва се формата на съществуващия клас
      - Без да се променя съществуващия (базовия) клас
      - Добавя се нов код

НАСЛЕДЯВАНЕ

- Наследяване
  - Отношение между класовете
  - Елемент на моделиране в софтуерните архитектури
- Класове: кога се разработват?
  - Най-късно: във фазата на проектиране чрез анализ на изискванията
  - Цел: още при 'анализ и дефиниция' класовете да бъдат идентифицирани

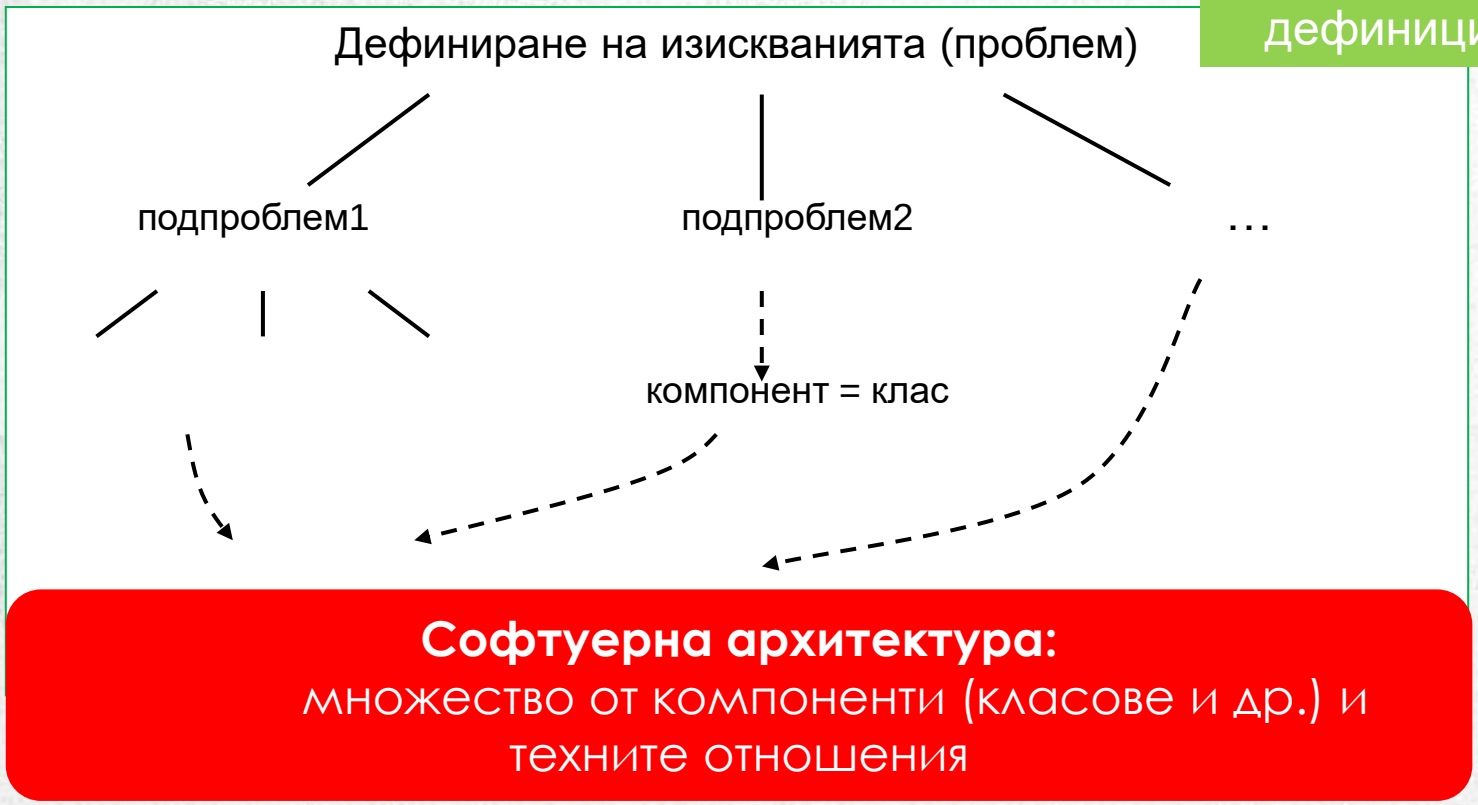
Софтуерен развой (фази):

- **Анализ и дефиниция:** дефиниране на изискванията
- **Проектиране:** архитектура на софтуера
- **Реализация:** програма

КАК ПОЛУЧАВАМЕ КЛАСОВЕТЕ?

Чрез декомпозиране на проблема

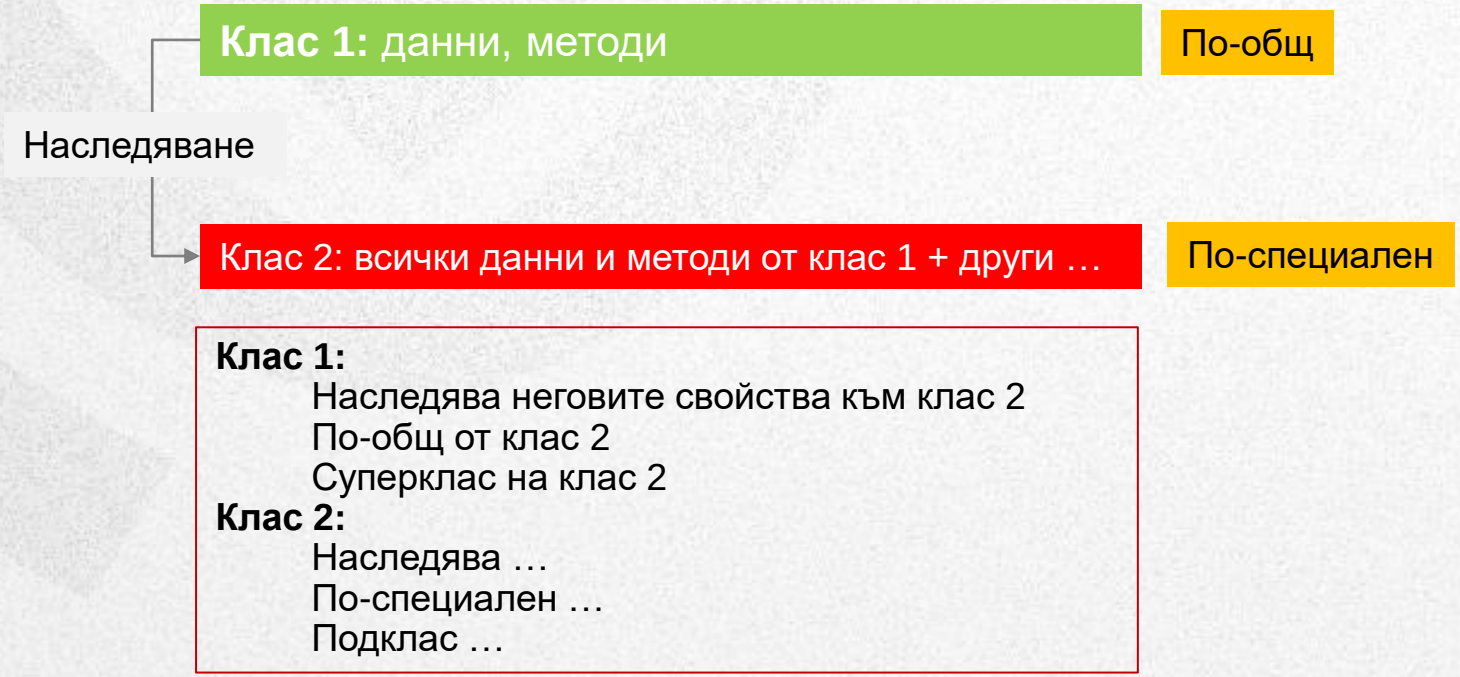
Фаза ‘анализ и дефиниция'



Софтуерна архитектура:
множество от компоненти (класове и др.) и техните отношения

ОТНОШЕНИЯ МЕЖДУ КЛАСОВЕ

Наследяване , асоциация, агрегация, …



Клас 1:
- Наследява неговите свойства към клас 2
- По-общ от клас 2
- Суперклас на клас 2

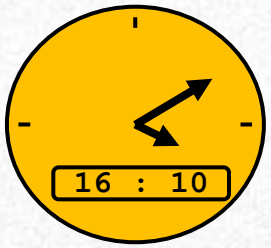
Клас 2:
- Наследява …
- По-специален …
- Подклас …

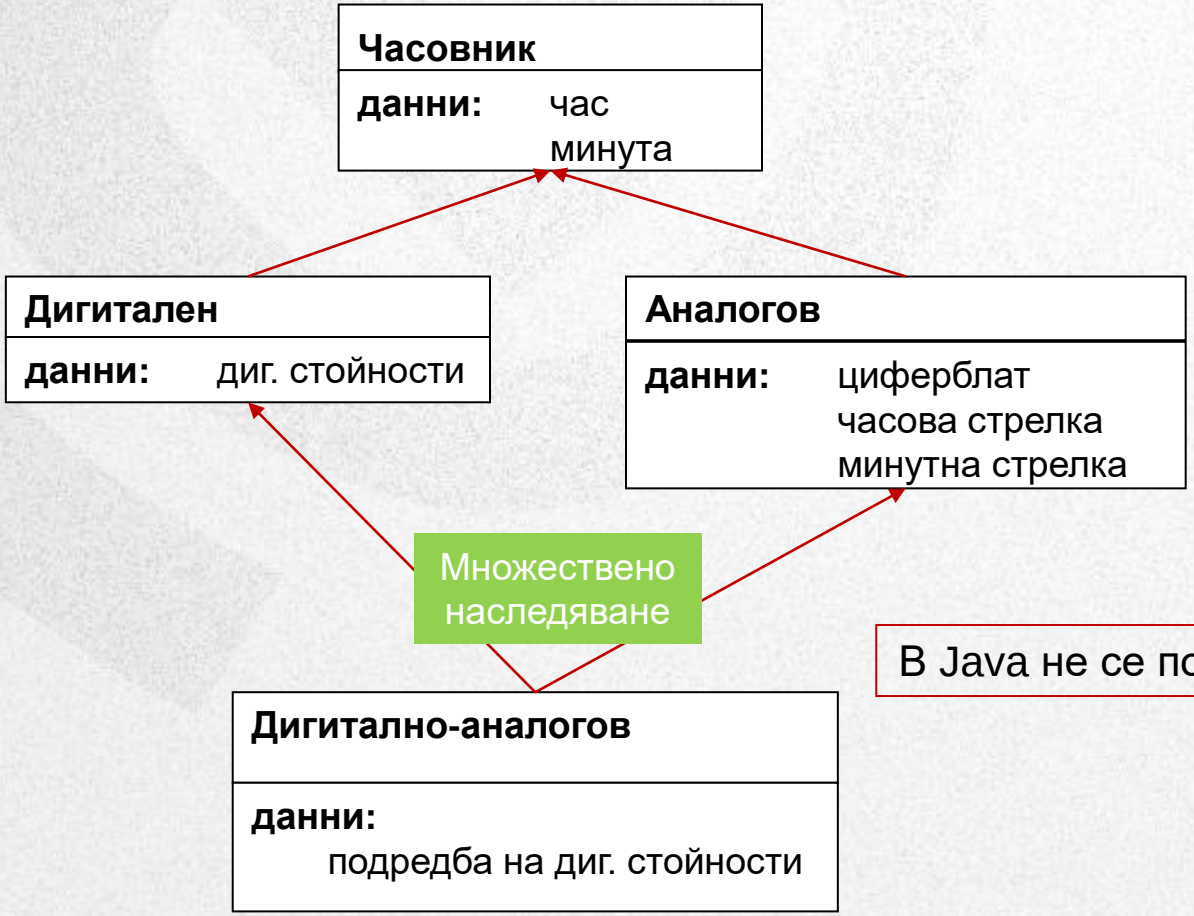
ПРИМЕР ЗА НАСЛЕДЯВАНЕ

Управление на недвижимо имущество:

нотация: UML (език за графично описание на софтуерни архитектури)

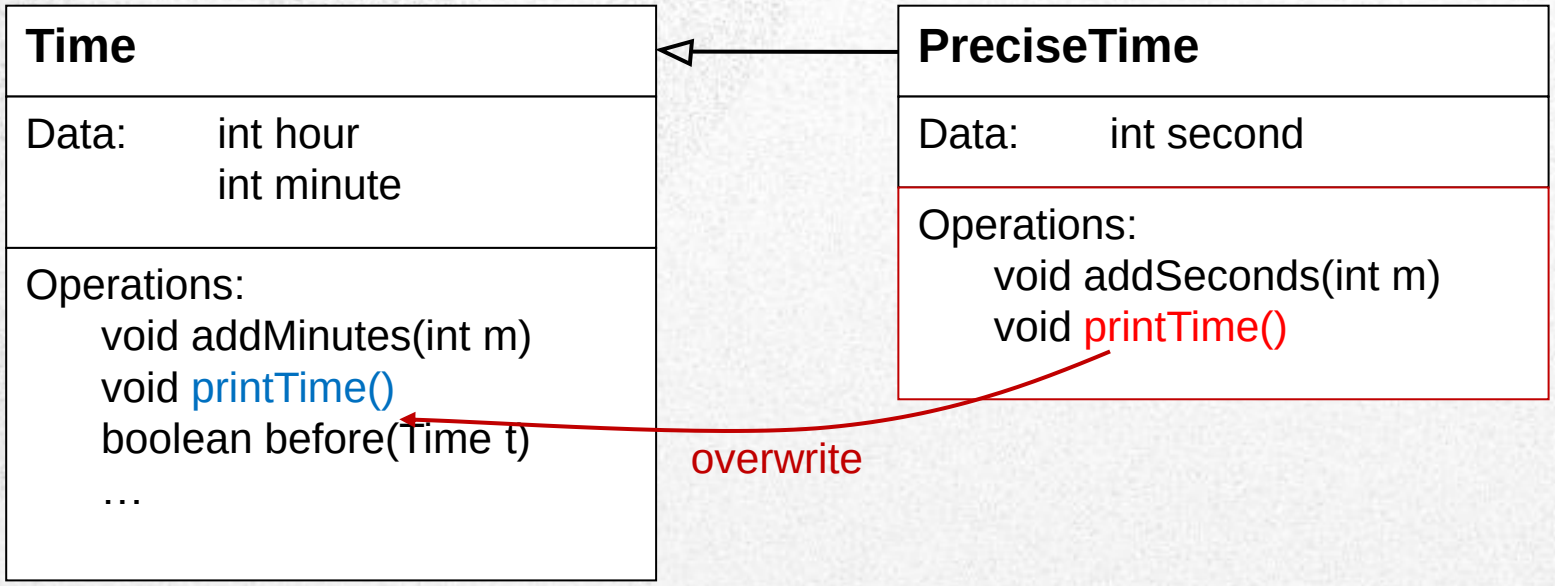
ПРИМЕР ЗА (МНОЖЕСТВЕНО) НАСЛЕДЯВАНЕ





В Java не се поддържа (но в C++ да)

ЧАСОВНИК: ТОЧНО ВРЕМЕ



СЪЗДАВАНЕ МЕТОДИ ЗА ПОДКЛАС

- При създаване на подкласове можем да специфицираме допълнителни данни и методи
- При дефиниране на методи в един подклас съществуват следните три възможности:
  - **Препокриване** на методи от суперкласа
    - Специфицираме методи със същата сигнатура като в суперкласа
  - **Наследяване** методи от суперкласа
    - Ако един метод не е препокрит, тогава автоматично се наследява
  - **Дефиниране** на нови методи
    - Специфицираме нов метод, който не съществува в суперкласа
    - Могат да бъдат използвани само за обектите на подкласа

СЪЗДАВАНЕ ДАННИ ЗА ПОДКЛАС

- Ситуацията при данните е по-различна:
  - **Никога** не могат да се **препокриват** данни
    - Новите данни "засенчват" тези от суперкласа
      - т.е. данните от суперкласа са налични, но не са достъпни
  - **Наследяване** данни от суперклас
    - Всички данни се наследяват автоматично
  - **Дефиниране** на нови данни
    - Представени са само в подкласа

НАСЛЕДЕН КЛАС

1 Коментар на наследения клас?

Базов клас (непроменен)

```
class Time {
   private int hour, minute;
   public Time() ...
   public void addMinutes(...)
   public void printTime()
}
```

```
class PreciseTime extends Time {

   private int second;

   public PreciseTime(...) ...

   public void addSeconds(int s)

   public void printTime()
}
```

Нова променлива

Нов конструктор

Нов метод

Препокриване

ВИДИМОСТ

1 Къде са видими hour, minute? само в клас 'Time'

```
class Time {
   private int hour, minute;
   ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

ВИДИМОСТ

1. Какво предизвиква използването на hour, minute в наследения клас?
2. Защо?

```
class Time {
   private int hour, minute;
   ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

Грешка!

ВИДИМОСТ

1 Решение? Видими във всички подкласове

```
class Time {
 protected int hour, minute;
   ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

ИЗПОЛЗВАНЕ: НАСЛЕДЕН КЛАС

```
class Time2 {

  public static void main(String[] args) {
    PreciseTime lunchtime = new PreciseTime(12, 1, 0);

    lunchtime.addMinutes(1);
    lunchtime.printTime();
    lunchtime.addSeconds(-61);
    lunchtime.printTime();
    lunchtime.addSeconds(1);
    lunchtime.printTime();
  }
}
```

Наследен

Нов

Препокрит

```
% java Time2
12:02:00PM
noon
12:01:00PM
```

БАЗОВ И НАСЛЕДЕН КЛАС

1. Как изглежда наследения (производен) клас за външния свят?

За външния свят:
- Новият клас има същия интерфейс, като този на базовия
- Евентуално някои допълнителни методи и полета

2. Какви са обектите на наследения (производен) клас?

Наследяването не е просто копиране на интерфейса на базовия клас:
- Обектите на производния клас съдържат в себе си подобект на базовия клас
- Този подобект – все едно, че е създаден обект на самия базов клас
- Отстрани, подобектът ще изглежда като обвит от обекта на наследения клас

ИНИЦИАЛИЗАЦИЯ НА БАЗОВ КЛАС

- От съществено значение е подобектът на базовия клас да бъде коректно инициализиран
- Само един начин, гарантиращ това …

1 Кой?

Извикване конструктора на базовия клас:
- Има необходимите права за извършване инициализация на базовия клас
- Java автоматично вмъква извиквания към конструктора по подразбиране на базовия клас към конструктора на производния клас

ПРИМЕР

```
class Plant {
      Plant() {
          System.out.println("Plant constructor");
      }
}

class Tree extends Plant {
     Tree() {
          System.out.println("Tree constructor");
     }
}

public class Elm extends Tree {
     Elm() {
          System.out.println("Elm constructor");
     }

     public static void main(String[ ] args) {
           Elm e = new Elm();
     }
}
```

1 Какъв резултат?

Plant constructor
Tree constructor
Elm constructor

ПРИМЕР: CHESS

```
class Game {
    Game(int i) {
        System.out.println("Game constructor");
    }
}
class BordGame extends Game {
   BordGame(int i) {

        System.out.println("BordGame constructor");
   }
}
public class Chess extends BordGame {
    Chess() {

        System.out.println("Chess constructor");
    }

    public static void main(String[ ] args) {
        Chess x = new Chess();
    }
}
```

1 Разлика с Elm?

Конструкторът на базовия клас е с параметри

2 Как ще му бъдат предадени параметрите?

super

ПРИМЕР: CHESS

```
class Game {
    Game(int i) {
        System.out.println("Game constructor");
    }
}
class BordGame extends Game {
   BordGame(int i) {
        super(i);
        System.out.println("BordGame constructor");
    }
}
public class Chess extends BordGame {
    Chess() {
        super(11);
        System.out.println("Chess constructor");
    }

    public static void main(String[ ] args) {
        Chess x = new Chess();
    }
}
```

1 Какъв резултат?

Game constructor
BordGame constructor
Chess constructor

ПРИМЕР: TIME

```
class Time {
   private int hour, minute;
   public Time (int h,int m) { hour = h; minute = m; }
   public void advanceMinutes (int m) {
    int totalMinutes = (60*hour + minute + m)%(24*60);
    if (totalMinutes < 0)
          totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60;
    minute = totalMinutes%60);
   }
   public void printTime () {
    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon  ");
    else {
      if      (hour == 0) System.out.print(12);
      else if (hour > 12) System.out.print(hour
      else                System.out.print(hour

      if (minute < 10) System.out.print(":0" +
      else             System.out.print(":" + m

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
  }
}
```

```
class PreciseTime extends Time {
   private int second;
   public PreciseTime (int h,int m, int s) {
      super(h, m)
      second = s;
   }
   public void advanceSeconds (int s) {
    int advMinutes = s / 60;
    second += s % 60;
    if (second < 0) {
      advMinute--;
      second += 60;
    }
    else if (second >= 60) {
      advMinute++;
      second -= 60;
    }
      advanceMinutes(advMinutes);
   }
}
```

ЯВНО ИЗВИКВАНЕ НА МЕТОДИ НА БАЗОВИЯ КЛАС

1 Възможно ли е super.super ? не

```
class Time {
  protected int hour, minute;
    . . .
  public void printTime () {…}
}
```

```
class PreciseTime extends Time {
  private int second;

  public void printTime () { …}

  public void printTimeShort () {
     super.printTime();
  }
}
```

БЕЗ НАСЛЕДЯВАНЕ

1 Може ли без наследяване? да

```
class Time { /* както досега */ }
```

```
class PreciseTime {
   private int hour, minute;
   private int second;
   public PreciseTime(...) ...
   public addMinutes(...)
   public void addSeconds(...)
   public void printTime()
}
```

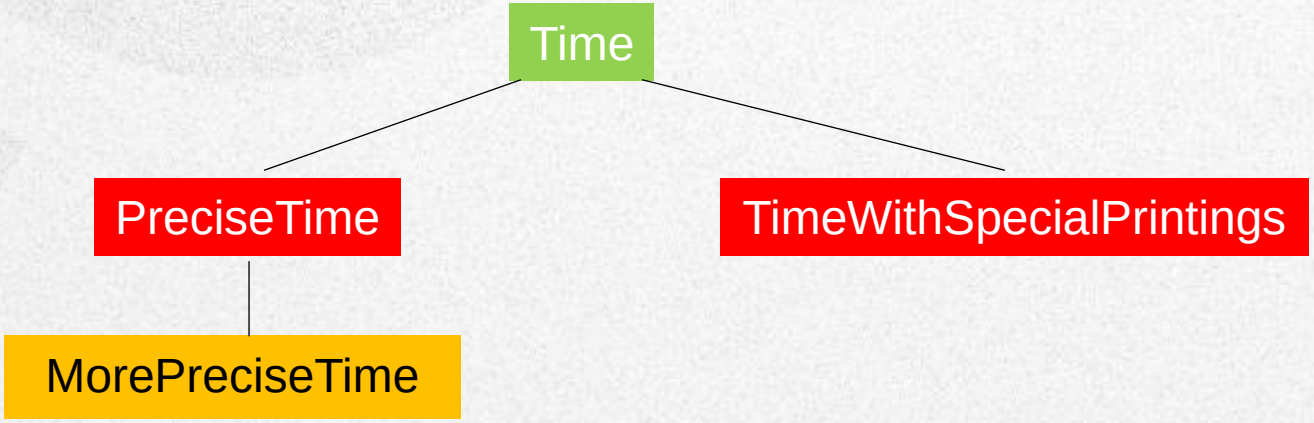
- Двете версии са еквивалентни! (можем без наследяване)
- По-дълги програми
  - Повтаряне на код:
    - Промените са по-сложни, ако кодът, който ще се променя е двойно по-голям
  - Не се отразяват логическите отношения между класовете

ОБОБЩЕНИЕ НА КОНТРОЛА НА ДОСТЪП

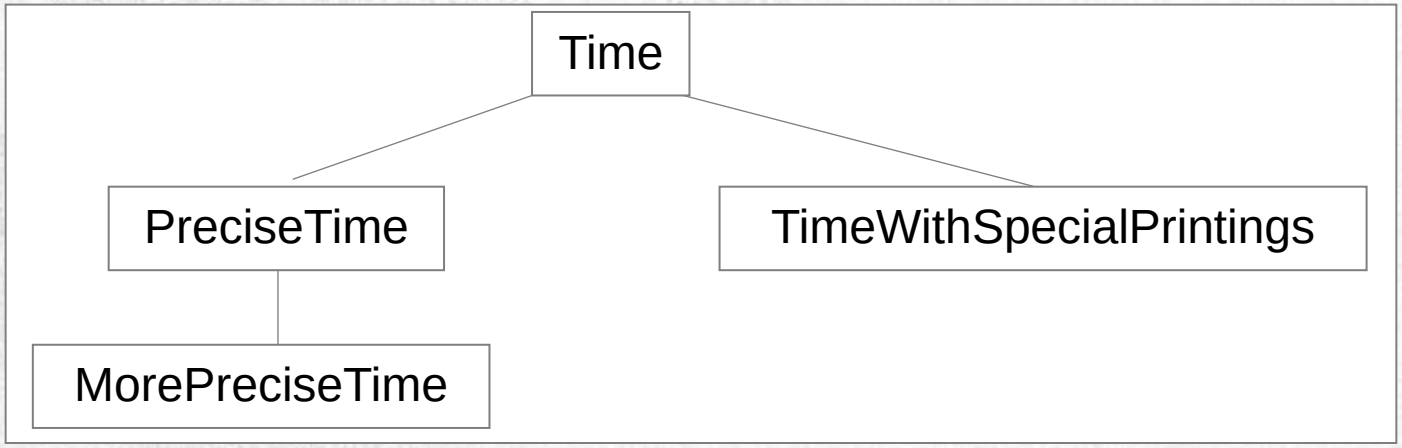
- Java доставя **четири нива** за контрол на достъпа до данни, методи и класове:
  - **public** достъп
    - Посредством методи от всички класове
  - **private** достъп
    - Само посредством методи от собствения клас
  - **protected** достъп
    - От собствения клас и неговите подкласове
  - **package** достъп
    - От всички методи в собствения пакет
    - Стандартна (по подразбиране) възможност

НАСЛЕДСТВЕНИ ЙЕРАРХИИ

```
class Time ...

class PreciseTime extends Time ...

class MorePreciseTime extens PreciseTime ...

class TimeWithSpecialPrintings extends Time ...
```



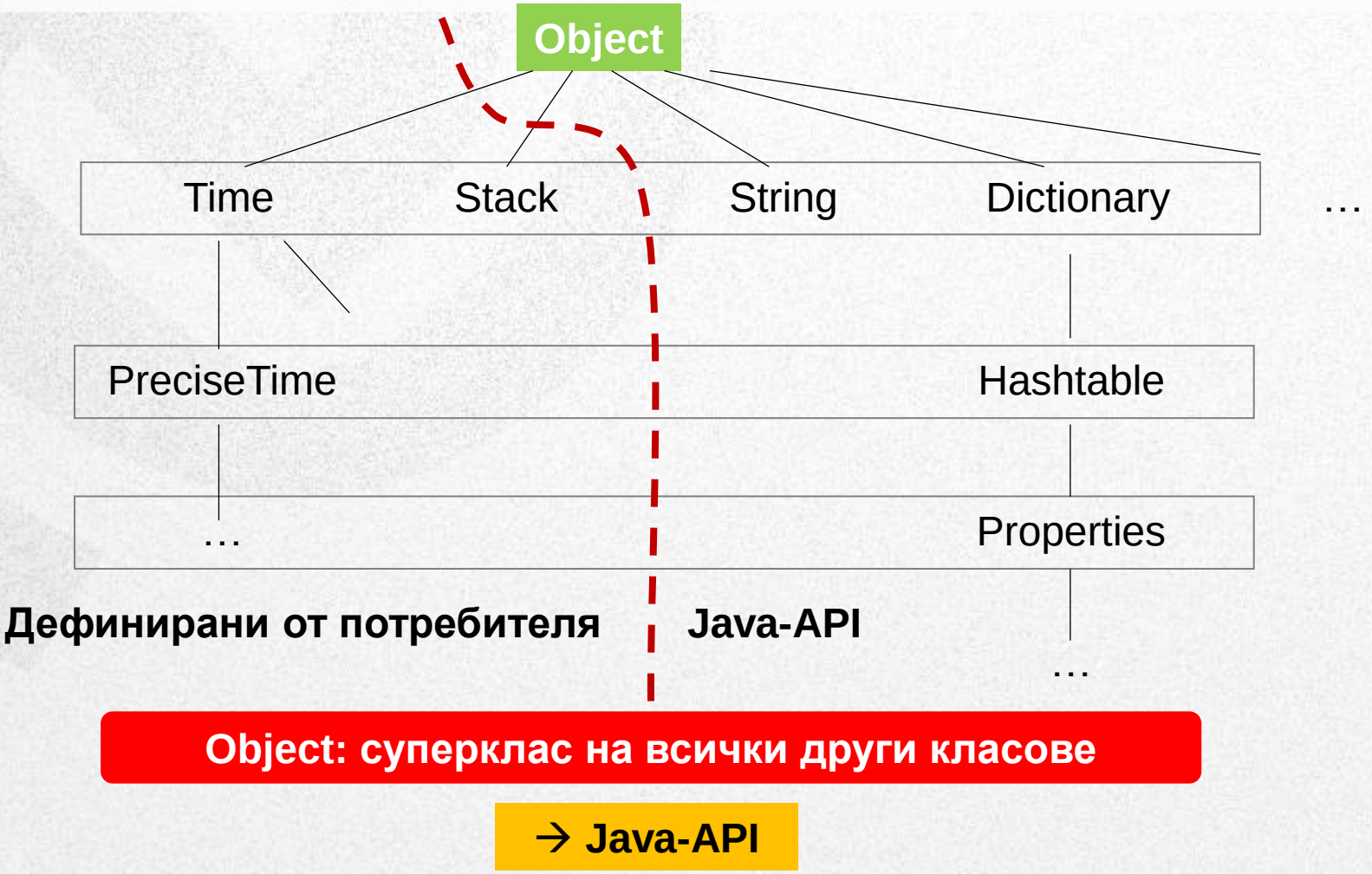
НАСЛЕДСТВЕНИ ЙЕРАРХИИ



Суперкласове на MorePreciseTime:
PreciseTime (директен суперклас)
Time

Подкласове на Time:
PreciseTime (директен подклас)
MorePreciseTime
TimeWithSpecialPrintings (директен подклас)

'OBJECT': СУПЕРКЛАС НА ВСИЧКИ КЛАСОВЕ



API-Class java.lang.Object

java.lang

Class Object

`java.lang.Object`

public class **Object**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class

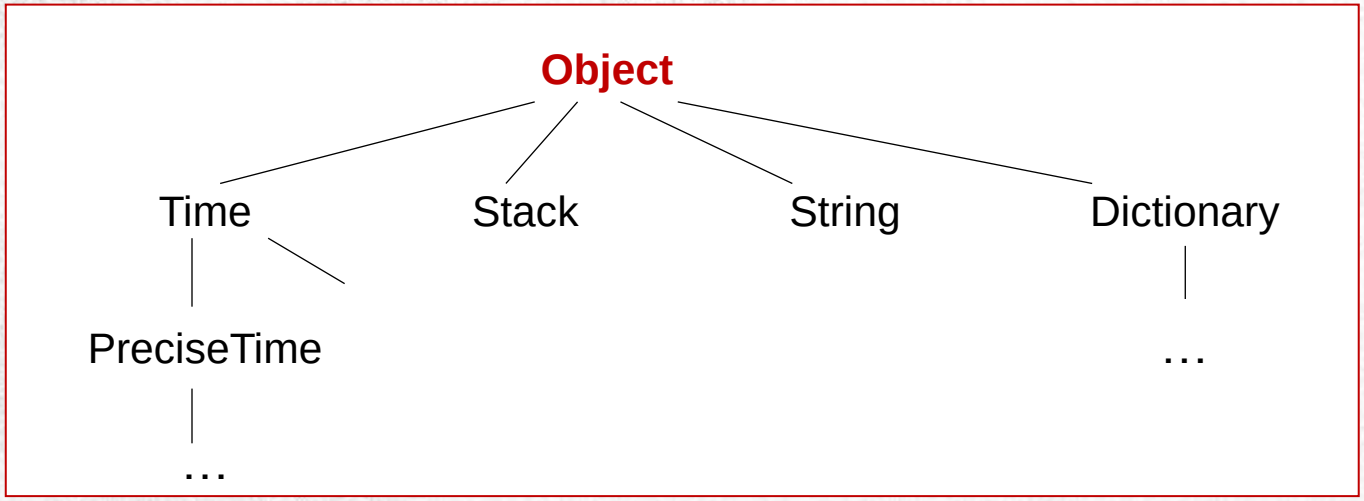
Constructor Summary

`Object``()`	

Method Summary

protected `Object`	**`clone`**`()` Creates and returns a copy of this object.
boolean	**`equals`**`(Object obj)` Indicates whether some other object is "equal to" this one.
protected void	**`finalize`**`()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class`	**`getClass`**`()` Returns the runtime class of an object.
int	**`hashCode`**`()` Returns a hash code value for the object.
void	**`notify`**`()` Wakes up a single thread that is waiting on this object's monitor.
void	**`notifyAll`**`()` Wakes up all threads that are waiting on this object's monitor.
`String`	**`toString`**`()` Returns a string representation of the object.
void	**`wait`**`()` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
void	**`wait`**`(long timeout)` Causes current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
void	**`wait`**`(long timeout, int nanos)` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

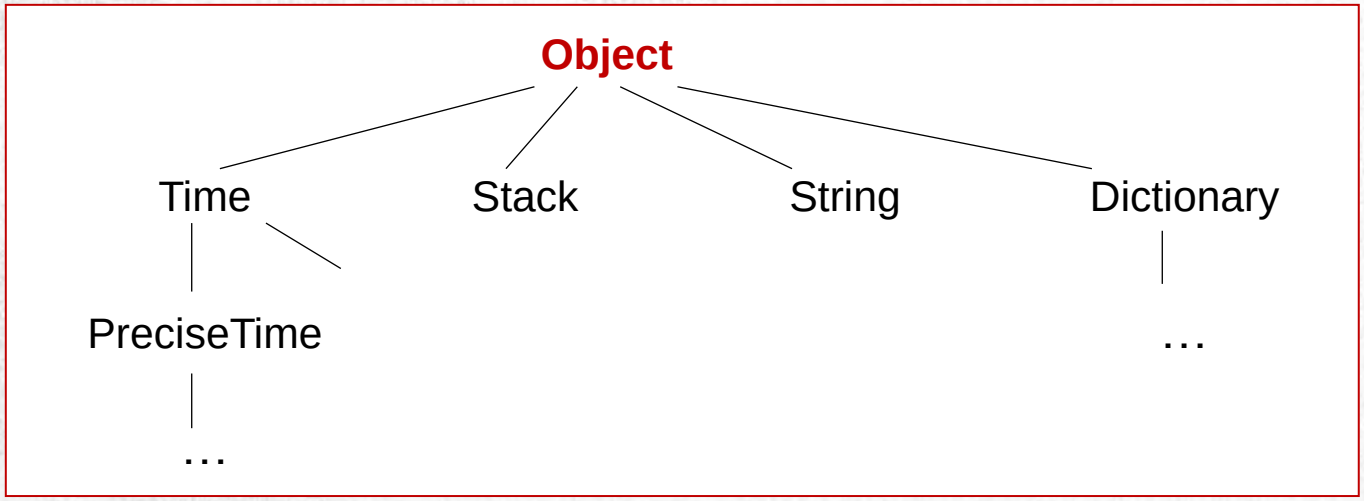
ПРАВИЛА ЗА СЪВМЕСТИМОСТ



Обектите на един подклас могат да стоят там, където са допустими обекти на суперкласовете.

```
Time t;
t = new PreciseTime(12, 10, 1);
t.printTime();
```

СЪВМЕСТИМОСТ С OBJECT



Извод: променливите на тип Object могат да притежават като стойност инстанции на произволни класове

```
Object o1, o2;
o1 = new PreciseTime(12, 10, 1);
o2 = new Stack(100);
```

ПОЛИМОРФИЗЪМ: ОСНОВНА ИДЕЯ

Полиморфизъм: "много форми на появяване"

```
Time t1;
PreciseTime t2;

t1 = new Time(12, 10);
t2 = new PreciseTime(0, 10, 1);
t1.printTime();
t2.printTime();
```

Полиморфизъм:
- Подобни операции с **една и съща** сигнатура (име, брой и тип на параметри)
- Поведението може да варира в зависимост от актуалния тип на обекта

Повторение: Overloading = едно и също име за методи с различни списъци на параметрите

ПОЛИМОРФИЗЪМ: ИЗБОР НА ПРАВИЛНИЯ МЕТОД

1. Кой избира: компилатор или интерпретатор?

```
Time t1;
PreciseTime t2;

t1 = new Time(12, 10);
t2 = new PreciseTime(0, 10, 1);
t1.printTime();
t2.printTime();
```

Изборът на оператор, зависим от целевия обект (напр. t1, t2)

ДИНАМИЧНО СВЪРЗВАНЕ

1. От клас Time или PreciseTime?

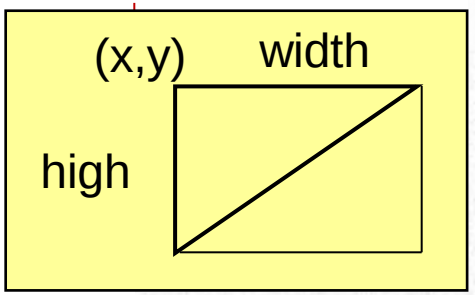
```
Time t;
if (readValue() == 'T')
   t = new Time(12, 10);
else
   t = new PreciseTime(12, 10, 1);
t.printTime();
```

Компилатор: не може да реши!

Така: избор на метод едва в run-time (по време на изпълнение на програмата)

АБСТРАКТНИ КЛАСОВЕ: СЪДЪРЖАТ МЕТОДИ БЕЗ РЕАЛИЗАЦИИ

```
abstract class Figure {
   protected int x, y, width, high;
   public void setx(int xNew) {
      x = xNew;
   }
   // setY, setWidth, setHigh
   public abstract float calculateSquare();
}
```



‚**calculateSquare**‘: метод без реализация

→ Реализацията трябва да се извърши в подкласа

Figure → правоъгълник, триъгълник, …

→ Не е разрешено създаване на инстанции на класа

→ **Смисъл**: логическо структуриране на множества от класове

МОТИВАЦИЯ ЗА АБСТРАКТНИ КЛАСОВЕ

- Когато разширяваме съществуващ клас имаме избора дали да предефинираме методи на суперкласа
- В определени случаи е желателно да заставим програмистите да редефинират определени методи
  - Напр. не съществува добър default (по подразбиране) за суперкласа
  - Само програмистът на подкласа знае каква е подходящата реализация на метода

ВИДОВЕ КЛАСОВЕ

- Абстрактен клас
  - За който не можем да създаваме обекти
  - Ключова дума abstract
  - Клас, който дефинира един абстрактен метод или който наследява абстрактен метод без да го препокрива
- Конкретен клас
  - Можем да създаваме обекти
  - Може да има обектни референции, типът на които са абстрактни класове

FINAL: ОГРАНИЧЕНИЕ НА ПОЛИМОРФИЗМА

- Методите не могат да бъдат предефинирани в подкласа
  → Идея: грижа за стабилна семантика (сигурност)

```
class Time {
   public final void addMinutes(...) {
      ...
   }
   public void printTime()
}
```

- Класовете не могат да бъдат разширявани напр. Java-API

```
public final class String;
```

МОТИВАЦИЯ ЗА FINAL

- Обратна на абстрактните класове логика
  - Избягваме други програмисти да създават подкласове или да препокриват методи
  - Примери:
    - public final class String { … }
    - public final boolean checkPassword(String password)

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “НАСЛЕДЯВАНЕ”

Distributed
@learning center