Classes and Objects in Java

الصفوف و الأغراض في الجافا

Dr. REEMA AL-KAMHA

مقدمة

- الجافا هي لغة برمجة غرضية التوجه، و هذا يعني أن برامج لغة الجافا مكونة من صفوف classes
- **يشكل مفهوم الصف الركن الأول في البرمجة غرضية التوجه**
- تستخدم الصفوف بلغة الجافا كأنواع، بحيث يمكن إنشاء أغراض من هذا الصف.
- **التغليف Encapsulation من أهم مبادئ البرمجة غرضية التوجه و تغليف حقول الصف أي جعلها خاصة private ، والوصول إليها فقط من خلال طرق عامة معينة public methods (setters, getters)**

مفهوم الصفوف classes

- إن تعريف صفٍّ class من الصفوف يعني تعريف قالب عام أو تصميم لأغراض من نفس النوع
- يمكن الحديث عن صف شخص Person، يعرف هذا الصف الصفات المشتركة التي تكون موجودة عند الأشخاص مثل الاسم، الجنس، الوظيفة، العمر و غيرها. ولا يعني ذلك أن جميع الأشخاص لهم نفس قيم الصفات المشتركة، بل يعني أن كل شخص يختلف عن الآخر بالقيم التي تأخذها صفة خاصة به.
- حجز الصف في الذاكرة يتم في وقت الترجمة وليس وقت التنفيذ لأن تصميم الشكل العام للأغراض لا يعني إنشاؤها
- إنشاء الأغراض يتم وقت التنفيذ

صف Person يمتلك الصفات التالية

الإسم:
الجنس:
الوظيفة:
العمر:

أغراض من الصف Person

الإسم: محمد	الإسم: روز	الإسم: أحمد	الإسم: ربيع
الجنس: ذكر	الجنس: أنثى	الجنس: ذكر	الجنس: ذكر
الوظيفة: طالب	الوظيفة: سكرتيرة	الوظيفة: طبيب	الوظيفة: مهندس
العمر: 21	العمر: 22	العمر: 34	العمر: 27

علاقة الأغراض objects بالصفوف classes

- إذا كنت تنوي إنشاء برنامج بسيط لحفظ معلومات عن الأشخاص، هل ستنشئ صف لكل شخص ؟!
  - ❖ طبعاً لا، بل تنشئ صف واحد فقط يمثل شخص، و تضع فيه الواصفات (الصفات) Attributes التي تريدها أن تكون موجودة عند كل شخص مثل الاسم، الجنس، الوظيفة، العمر، وغيرها.
  - ❖ ثم تنشئ من الصف Person أغراض قدر ما شئت، و عندها يصبح كل غرض من هذا الصف عبارة عن شخص له معلوماته الخاصة.
- يمكن تشبيه الفرق بين الصف class والغرض object كشركة سيارات تعطي تصميم معين لطراز ما، فالتصميم يمثل تعريف الصف، إذ لا يمكن تصنيع سيارات قبل تصميمها بمعنى آخر يتم تصميم السيارة قبل التنفيذ في وقت التصميم أوالترجمة.
- أما بناء كل سيارة من هذا الطراز يمثل إنشاء غرض object واحد يتبع لنفس الطراز، ويتم ذلك في وقت التنفيذ.

مفهوم الصفوف classes

Classes are

- a collection of methods and data
- a blueprint used to construct many objects
- a great way to partition a software system
- A way to implement any type
- A **type** defines a set of values, and the allowable operations on those values

الصفوف هي:

- مجموعة منظمة و مغلفة من البيانات (data) و الطرق (الدوال) (methods) تطبق على هذه البيانات
- قالب عام يستخدم لإنشاء عدة أغراض
- طريقة لإنشاء نوع جديد
  - النوع type يعرف مجموعة من القيم و العمليات المسموح إجرائها على هذه القيم

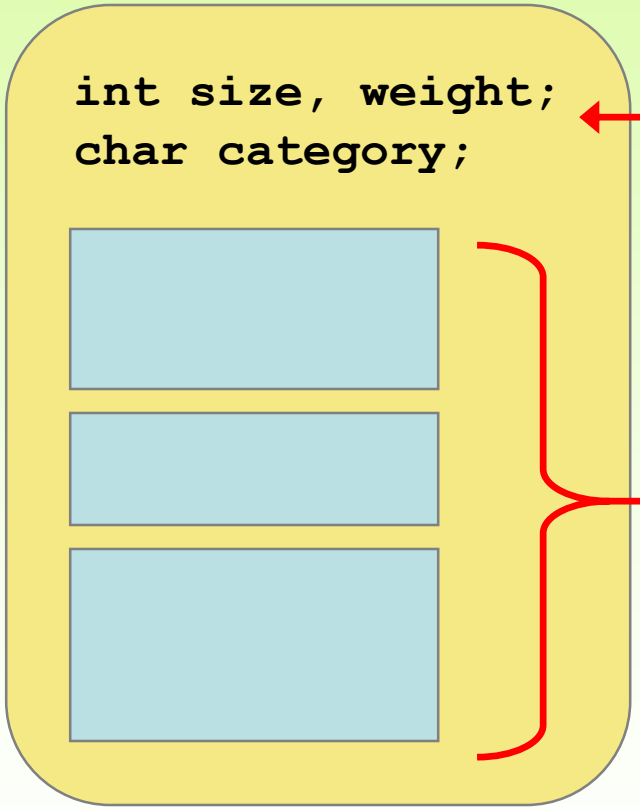
ملاحظة: قد يطلق على بيانات الصف بحقول (Fields) أو واصفات (Attributes) أو متغيرات (Variables)

Examples of Classes

Class	الحقول	الطرق
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

Classes

- A class can contain data declarations and method declarations



Data declarations

التصريح عن بيانات (حقول، واصفات، متغيرات) الصف

Method declarations

التصريح عن طرق الصف

Classes

- The values of the data define the state of an object created from the class
- The functionality of the methods define the behaviors of the object
- Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

- تعين قيم البيانات (قيم الحقول، قيم الواصفات، قيم المتغيرات) **حالة** state الغرض المنشأ من الصف
- تعين الطرق methods **سلوك** behavior الغرض المنشأ من الصف
- يوجد أنواع خاصة من الطرق في الصف تدعى بواني، و التي تستدعى لإنشاء أغراض من الصف.

الشكل العام للصف

يحتوي الصف على مجموعة من الأعضاء:

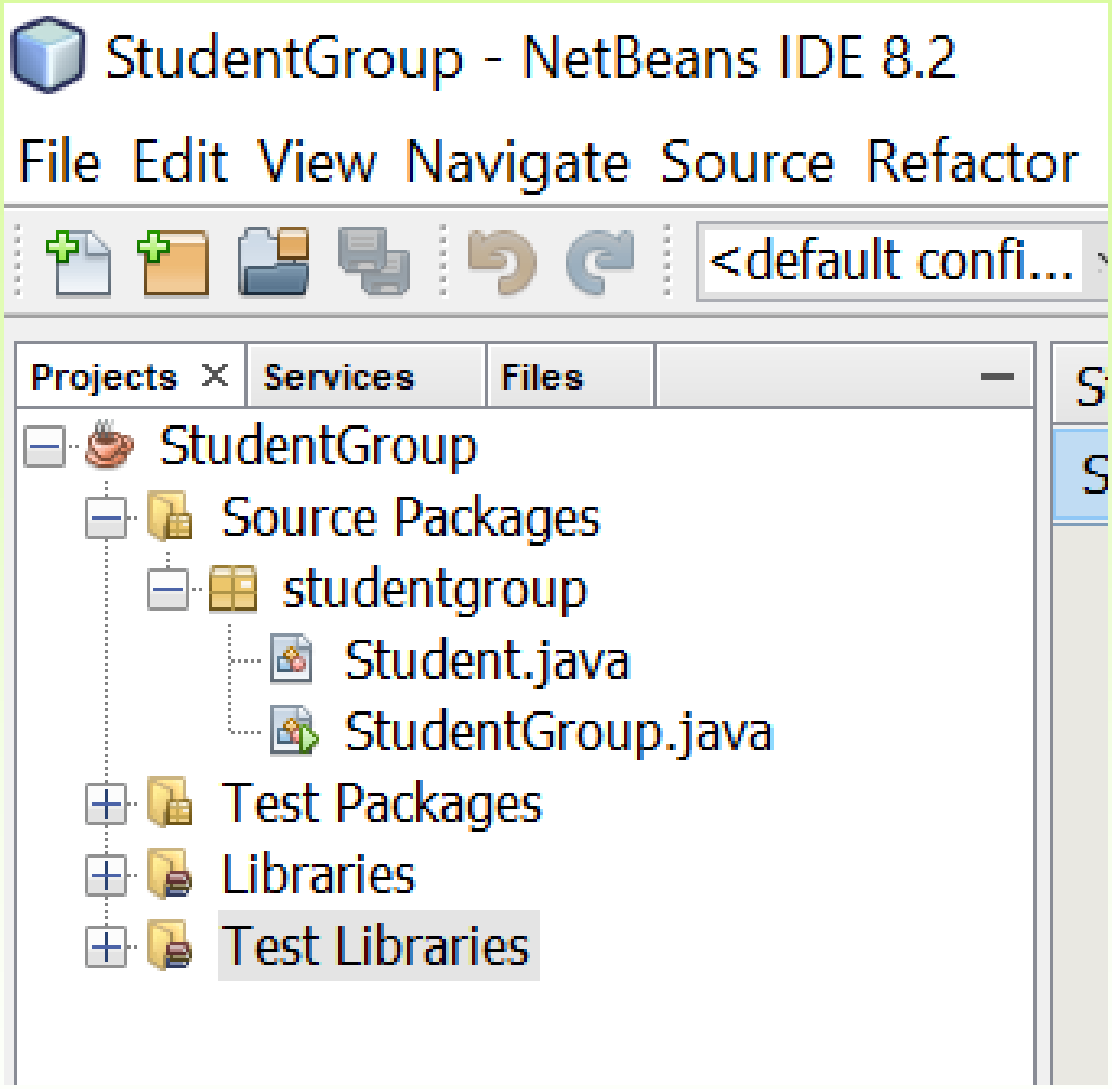
- بيانات Data (حقول Fields ، واصفات Attributes، متغيرات Variables) تخزن حالة الأغراض
- البواني Constructors لإنشاء أغراض من الصف (أي نوعها صف)
- الطرق Methods بعض الطرق لتعديل حالة الأغراض و أخرى للوصول إلى حالة الأغراض (أي بعضها يعدل على قيم الحقول للغرض و أخرى تسمح بالوصول إلى قيم الحقول للغرض)

لتوضيح كيفية كتابة الصف في الجافا سنبدأ بالمثال التالي:

اكتب برنامجا بلغة الجافا يقوم بما يلي:

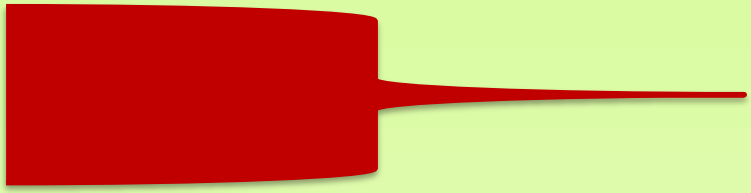
تعريف صف Student يحتوي المعلومات التالية:

- الحقول: اسم الطالب name، العمر age، وعدد الطلابnumber الذين تم إنشاؤهم
- الطرق:
  - باني يستقبل اسم الطالب و ينشئ غرض من الصف Student
  - باني يستقبل اسم الطالب، و عمره و ينشئ غرض من الصف Student
  - طريقة تعدل اسم الطالب
  - طريقة تعدل عمر الطالب بشرط أن يكون عمره أكثر من خمس سنوات
  - طريقة للحصول على اسم الطالب
  - طريقة للحصول على عمر الطالب
  - طريقة للحصول على عدد الطلاب الذين تم إنشاؤهم من الصف Student
- تعريف صف StudentGroup يقوم بما يلي:
  - ❖ إنشاء طالب اسمه Sami و عمره 10
  - ❖ إنشاء طالب اسمه Rami و عمره 12
  - ❖ طباعة عدد الطلاب الذين تم انشاؤهم
  - ❖ تعديل عمر Rami ليصبح 15
  - ❖ طباعة المعلومات المتعلقة بكل طالب

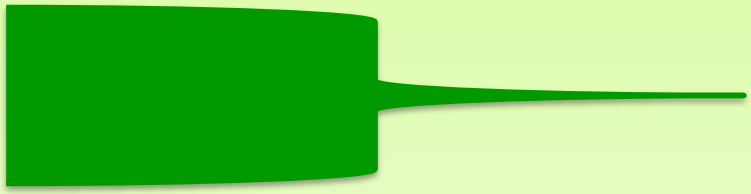

StudentGroup - NetBeans IDE 8.2
File Edit View Navigate Source Refactor
<default confi...
Projects
Services
Files
StudentGroup
Source Packages
studentgroup
Student.java
StudentGroup.java
Test Packages
Libraries
Test Libraries

```
package studentgroup;
public class Student {
    private String  name;
    private int age;
    static int number;
    public Student(String name){
        this.name=name;
        number++;
    }
    public Student(String name, int age){
        this.name=name;
        this.age=age;
        number++;
    }
    public void setName(String name){
        this.name=name;
    }
    public void setAge(int age){
        if(age>5)
          this.age=age;
    }
    public String getName(){
        return name;
    }
    public int getAge(){
        return age;
    }
    public static int getNumber(){
        return number;
    }
 }
```

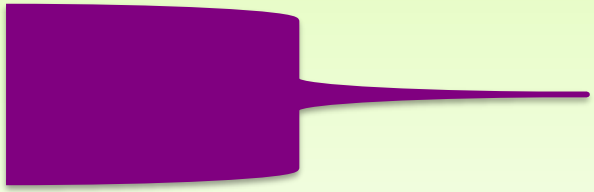
```
package studentgroup;
public class Student {
    private String  name;
    private int age;
    static int number;
    public Student(String name){
        this.name=name;
        number++;
    }
    public Student(String name, int age) {
        this.name=name;
        this.age=age;
        number++;
    }
    public void setName(String name){
        this.name=name;
    }
    public void setAge(int age){
        if(age>5)
            this.age=age;
    }
```



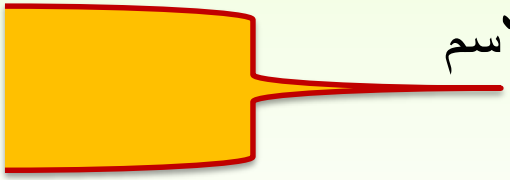
حقول الصف



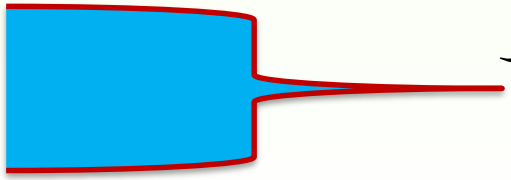
الباني الأول



الباني الثاني

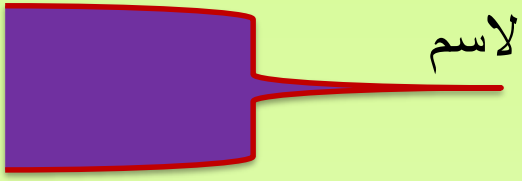


طريقة تسمح بتعديل الاسم

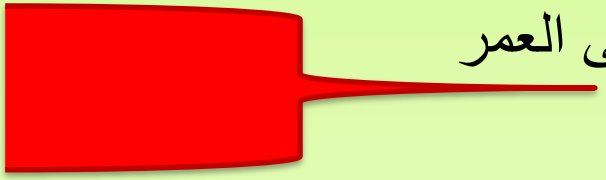


طريقة تسمح بتعديل العمر

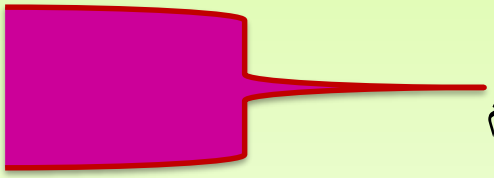
```
    public String getName(){
        return name;
    }
    public int getAge(){
        return age;
    }
    public static int getNumber(){
        return number;
    }
}
```



طريقة تسمح بالحصول على الاسم



طريقة تسمح بالحصول على العمر



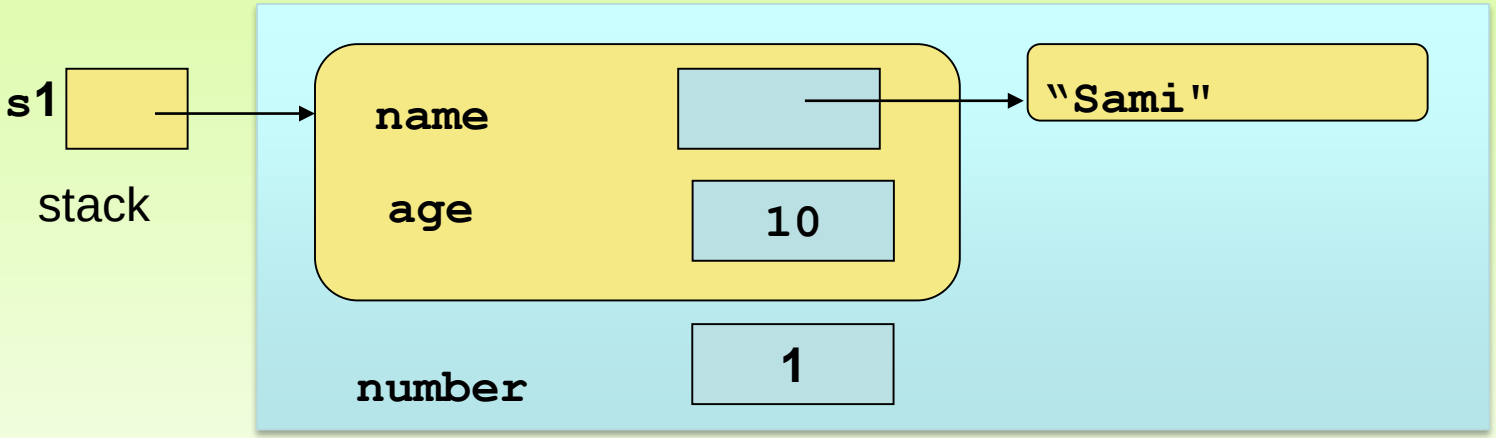
طريقة تسمح بالحصول على عدد الطلاب الذين تم إنشاؤهم من الصف

```
package studentgroup;
public class StudentGroup {
    public static void main(String[] args) {
     System.out.println("the number of students="+Student.getNumber());
     Student s1=new Student("Sami", 10);// إنشاء الطالب سامي و عمره عشر سنوات
     System.out.println("the number of students="+Student.getNumber());
     Student s2=new Student("Rami",12);// إنشاء الطالب رامي و عمره 12 سنة
     System.out.println("the number of students="+Student.getNumber());
      //طباعة أعداد الطلاب الذين تم إنشاؤهم
     s2.setAge(15);// تعديل عمر الطالب رامي
     System.out.println("Student1 name= "+s1.getName()+" Age="+s1.getAge());
      // طباعة معلومات الطالب الأول
     System.out.println("Student2 name= "+s2.getName()+" Age="+s2.getAge());
     // طباعة معلومات الطالب الثاني
    }
}
```

- لاحظ أنه بعد إضافة الصف **Student**، أصبح لدينا نوع بيانات جديد يمكن تعريف متغيرات منه (تسمى متغيرات مرجعية **Reference Types**) و إنشاء أغراض جديدة.
- المعامل **new** لإنشاء غرض جديد من الصف **Student**.
- المتغير **s1** والذي هو من النوع **Student** هو متغير مرجعي، يحوي عنوان الغرض في الذاكرة.

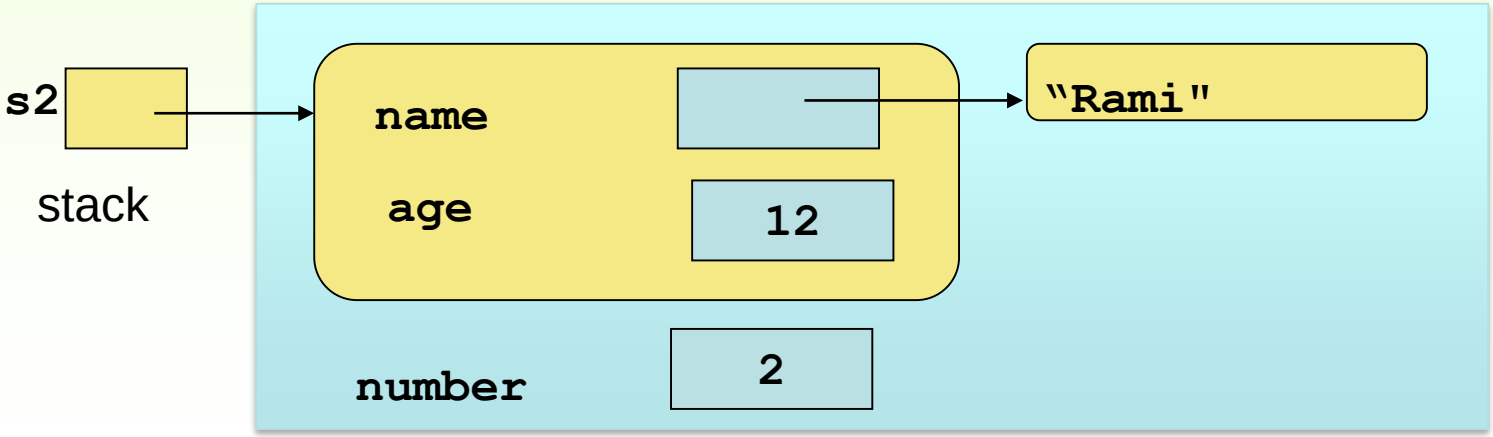
heap

Student s1=new Student("Sami", 10);



heap

Student s2=new Student("Rami",12);

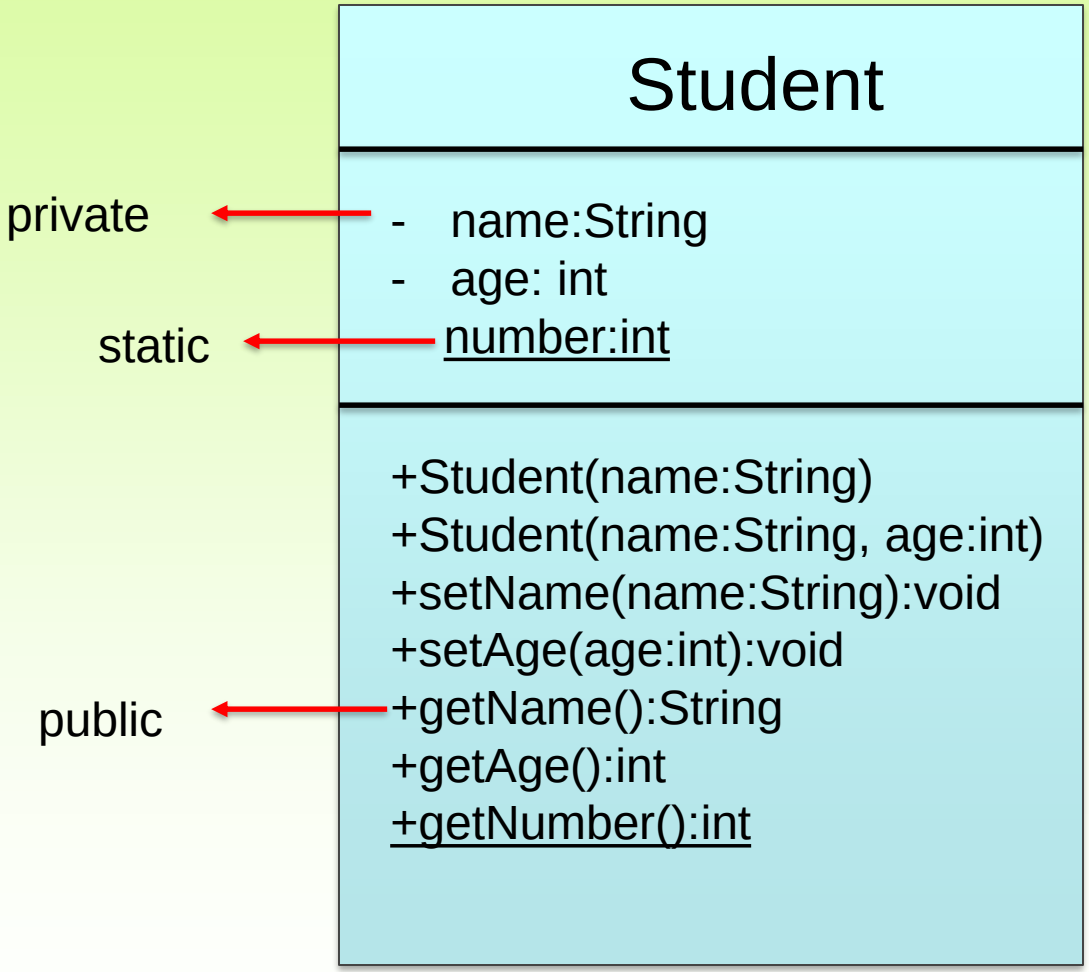


heap

التنفيذ

- the number of students=0
- the number of students=1
- the number of students=2
- Student1 name= Sami Age=10
- Student2 name= Rami Age=15

تمثيل الصف Student باستخدام لغة UML



public level	+
protected level	#
private level	-
static	يتم رسم خط تحتها

ملاحظات حول الصف Student

- نلاحظ أن الصف Student نوعه عام public، وهذا يعني أننا نستطيع تعريف أغراض objects منه في حزم packages أخرى غير الحزمة التي تحتويه، إضافة للحزمة التي تحتويه.
- الصف Student يحوي حقول هي name و age، وهذا يعني أنه كلما أنشئنا غرض object من هذا الصف، فإن الغرض سيملك نسخة خاصة من هذين الحقلين
- كل **حقول الصف تظل موجودة في الذاكرة طالما أن الغرض موجود**
- **لاحظ أنه بعد إضافة الصف Student، أصبح لدينا نمط بيانات جديد يمكن تعريف متغيرات منه و إنشاء أغراض جديدة.**

ملاحظات حول الصف Student

- حقول الصف خاصة private أي لا يمكن تغيير قيمتها من خارج الصف مباشرة، و إنما يتم ذلك عن طريق طرق عامة set و get المعرفتين لكل حقل ضمن الصف وهما من النوع public
  - الطرق setName, setAge هي طرق عامة public تسمح بتعديل قيم الحقول الخاصة name, age private للغرض الذي يحوي هذه الطرق، ويكون الارجاع فيها من النوع void (أي لا ترجع شيئا)
  - الطرق getName, getAge هي طرق عامة public تسمح بإرجاع قيم الحقول الخاصة name, age private للغرض الذي يحوي هذه الطرق، يكون الارجاع فيها من نوع الحقل المراد معرفة قيمته

ملاحظات حول الصف Student

- نلاحظ وجود بانيين constructors، وهو طريقة لا ترجع أي قيمة و يكون اسمها مطابق لاسم الصف و تستدعى عند إنشاء غرض من الصف
- يمكن أن يحوي الصف عدة بواني يستدعي المبرمج أحدها عند إنشاء الغرض وهذا ما يعرف اصطلاحا بـالتحميل الزائد للطرق overload
- استخدمنا كلمة المحجوزة this للإشارة إلى الغرض الحالي من الصف الذي نعمل ضمنه
- يجب أن يكون الباني من النوع public لأنه يستدعى دائما من خارج الصف
- لم نعرف هادما destructor للصف لأن garbage collector تحمل عبء هدم الغرض وكل ما يحتويه من حجز ديناميكي

ملاحظات حول الصف Student

- في الصف Student عرفنا الحقل number على أنه static، وهو يمثل عدد الأغراض (الطلاب) التي يتم إنشاؤها من الصف في البرنامج، فبالتالي هذا الحقل غير خاص بغرض معين و إنما يعتبر خاص بالصف Student و بالتالي لا يجوز أن يحوي كل غرض نسخه منه لأن هذا يمثل هدرا في الذاكرة، و إنما يكون الحقل نفسه مشتركا بين جميع الأغراض من النوع Student و نصل إليه من خارج الصف Student عن طريق اسم الصف(Student.number)، كما يمكن أيضا الوصول إليه عن طريق الغرض.
- كما أن الحقل من النوع static ليس حكرا على غرض معين، كذلك الطريقة الساكنة static methodعبارة عن طريقة تستدعى من اسم الصف ولا تخص غرض معين، كما أن الـ static method لا تستطيع التحكم إلا بالحقول من النوع static

بعض التوجيهات التي تخص كتابة الصف

- اجعل حقول الصف متحولات خاصه، بحيث يمكن الوصول إليها من داخل الصف فقط و لا يمكن الوصول لها من صفوف أخرى. وفي حال أردنا قراءة قيمها أو التعديل عليها نستطيع فعل ذلك عن طريق طرق عامة ضمن الصف
- اجعل معظم الطرق عامة
- ضع كل صف ضمن ملف

ملاحظات حول الحقول في الصف

- يمكن للصف أن يحوي مجموعة (قد تكون خالية) من الحقول
- يمكن أن تعطى حقول الصف قيما ابتدائية، و يمكن أن تترك بدون قيم ابتدائية.
في الحالة الأخيرة يقوم المترجم بوضع قيم افتراضية للحقول بحسب نوع الحقل

مثال: يصرح الصف التالي عن مجموعة من الحقول. أسندت لبعض هذه الحقول قيم (مثلا d=1.3).
لاحظ أن الحقل i لم تسند له قيمة، و بالتالي سيقوم مترجم اللغة بوضع قيمة افتراضية لهذا الحقل

```
class s{
int i;
Double d=1.3;
Boolean b=true;
}
```

Type	Default Value
boolean	False
byte	(byte) 0
short	(short)0
int	0
long	0L
char	\u 000
float	0.0f
double	0.0d
object information	null

محددات الوصول للصف Class Modifiers

- **المحدد الافتراضي package access**

إذا لم يوضع قبل اسم الصف أي محدد وصول صريح فإن الصف يتبع المحدد الافتراضي. وهذا يعني أنه يمكن الوصول للصف من الصفوف التي ضمن الحزمة. أي لا يمكن إنشاء أغراض (objects) منه إلا ضمن الصفوف التي تتشارك معه في نفس الحزمة.

- **المحدد الوصول العام public**

إذا وضع المحدد public قبل اسم الصف فإنه يمكن الوصول إلى هذا الصف أينما كان موجود على الحاسوب (سواء من نفس الحزمة الموجود فيها الصف أو من صفوف في حزم أخرى). أي يمكن التصريح عن أغراض منه، أما مكوناته فتخضع لمحددات الوصول الخاصة بها

ملاحظة: المحدد الخاص private **لا يمكن** وضعه قبل اسم الصف حيث أنه سيصبح الصف خاصاً، وبالتالي لا يمكن إنشاء أغراض منه، ولا يمكن الاستفادة منه بأي شكل.

ولكن يمكن وضعه قبل مكونات الصف (الحقول و الطرق)، وإذا وضع قبل المكونات فلا يمكن الوصول إلى هذه المكونات إلا ضمن الصف ذاته **أي لا يمكن الوصول اليهم من خارج الصف أبداً سواء من نفس الحزمه أم لا**

Class Modifiers محددات الوصول للصف

- **المحدد المحمى protected**

  - **لا يستخدم قبل اسم الصف**
  - يستخدم مع مكونات الصف (الحقول و الطرق) فقط، حيث يكون المكون الذي له محدد وصول محمي عاما في الصف ذاته وضمن الصفوف التي ترتبط معه بعلاقة وراثة والصفوف الموجودة معه في نفس الحزمة، و خاصا مع صفوف الحزم الأخرى التي لا ترتبط معه بعلاقة وراثة

محددات الوصول بالنسبة لحقول و طرق الصف

محدد الوصول	الوظيفة
public	إذا ظهر في بداية عبارة التصريح عن حقل أو طريقة فإنه يجعلهما عامين، أي يمكن الوصول إليهما من أي صف آخر
private	إذا ظهر في بداية عبارة التصريح عن حقل أو طريقة فإنه يجعلهما خاصين، ولا يمكن الوصول إليهما من خارج الصف الذي تم التصريح فيه
protected	إذا ظهر في بداية عبارة التصريح عن حقل أو طريقة فإنه يجعلهما محميين، أي يمكن الوصول إليهما من صفوف الأبناء و الصفوف المنتمية إلى الحزمة نفسها
الافتراضي package access	يجعل هذا المحدد الحقول و الطرق عامة بالنسبة للصفوف المنتمية إلى الحزمة نفسها، و لكنها خاصة بالنسبة للصفوف المنتمية إلى مكتبات أخرى. لايوجد كلمة محجوزة لهذا المحدد إذ يكفي أن لا تضع أيا من المحددات السابقة

Visibility Modifiers

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**


Package p1
class C1
public int a;
protected int b;
int c;
private int d;
class C2
C1 o = new C1( );
o.a; ✓ يمكن الوصول له
o.b; ✓ يمكن الوصول له
o.c; ✓ يمكن الوصول له
o.d; ✗
class C3 extends C1
a; ✓ يمكن الوصول له
b; ✓ يمكن الوصول له
c; ✓ يمكن الوصول له
d; ✗
Package p2
class C4
C1 o = new C1( );
o.a; ✓ يمكن الوصول له
o.b; ✗
o.c; ✗
o.d; ✗
class C5 extends C1
a; ✓ يمكن الوصول له
b; ✓ يمكن الوصول له
c; ✗
d; ✗
الكائن o يملك نسخة من كل شيئ موجود في الكلاس C1 و نستطيع الوصول إلى المتغيّر a فقط لأنه public
الكلاس C5 يرث كل شيئ موجود في الكلاس C1 و نستطيع الوصول إلى جميع المتغيّرات الموجودة ما عدا المتغيّر d لأنه private و المتغيّر c لأنه خارج الـ package الموجود فيها يعتبر أيضاً private
الكائن o يملك نسخة من كل شيئ موجود في الكلاس C1 و نستطيع الوصول إلى جميع المتغيّرات الموجودة ما عدا المتغيّر d لأنه private
الكلاس C3 يرث كل شيئ موجود في الكلاس C1 و نستطيع الوصول إلى جميع المتغيّرات الموجودة ما عدا المتغيّر d لأنه private

مثال

```
public class Run{
   int   x;
   double  y;
}
```

- بما أن محدد الوصول للصف Run هو public ، فإنه يمكن إنشاء أغراض objects منه من أي مكان (سواء من صفوف ضمن الحزمة الموجود فيها الصف Run أو من صفوف واقعة خارج الحزمة الموجود فيها الصف Run).
- لكن محددات الوصول إلى حقوله x و y هي المحددات الافتراضية، أي يمكن الوصول لهذه الحقول من قبل الصفوف الموجودة في نفس الحزمة الموجود فيها الصف Run، ولا يمكن الوصول لهم من صفوف واقعة خارج الحزمة الموجود فيها الصف Run

مثال

```
public class Tur{
    public int  x;
    private double  y;
}
```

- لإنشاء غرض object من الصف Tur، نكتب Tur t=new Tur(); و ذلك من أي صف سواء ضمن نفس الحزمة أو خارجها لأن الصف عام أي يمكن الوصول له من أي مكان داخل و خارج الحزمة.
- يمكن الوصول إلى الحقل x عن طريق كتابة t.x=5; و ذلك لأن الحقل عام يمكن الوصول له من أي مكان داخل و خارج الحزمة.
- الحقل y هو حقل خاص أي يمكن الوصول له فقط ضمن الصف Tur، ولكن لايمكن كتابة t.y=6.4; ضمن أي صف لا داخل الحزمة الموجود فيها الصف Tur و لا أي حزمة أخرى.

بما أن المتغيرات لا يمكن الوصول اليها الا من ضمن الصف نفسه كيف يمكننا الوصول اليها لتعديل قيمهم او طباعتها؟

يتم ذلك عن طريق تعريف طرق (دوال) عامه (طريقة تقوم باعطاء قيمة للمتغير وطريقة تقوم بارجاع قيمته)

```
public class Tun{
    public int x;
    private double y;
    public void set(int y){
        this.y=y;
    }
    public double get(){
        return y;
    }
}
```

منه سنقوم بما يلي في الصف الرئيسي:

```
Tun t=new Toto();
t.x=5;
t.set(6.5);
```

للوصول الى قيمة y لطباعتها مثلا يمكننا القيام بالتالي:

```
System.out.print(t.get());
```

```
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

public class C3 {
  cannot access C1;
  can access C2;
}
```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

ملاحظات حول static

- يمكن الوصول للحقول و الطرق الـ static من اسم الصف، و من اسم أي غرض من هذا الصف، ولكن العرف السائد أن يتم الوصول إليها من اسم الصف حصرا
- لا يمكن تعريف متغيرات من النوع static ضمن طريقة ما، و المكان الوحيد المسموح فيه استخدام الـ static هو بين حقول الصف
- يمكن إعطاء قيمة ابتدائية للحقل الـ static عند تعريفه مباشرة، و لكن هذه القيمة تعطى له عند ترجمة الصف و ليس عند إنشاء غرض، و بالتالي فهو يأخذها مرة واحدة فقط.

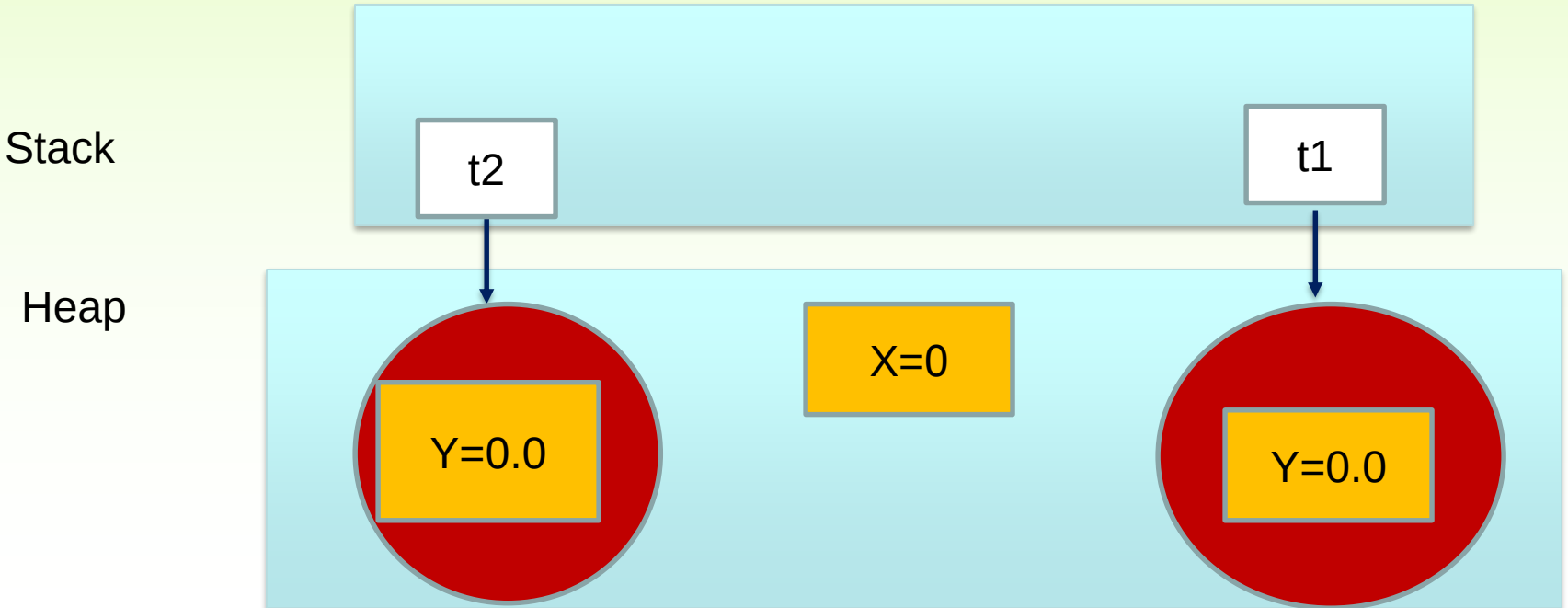
مثال

```
class Toto{
   static int x;
   double y;
}
```

ولننشئ منه الغرضين التاليين:

```
Toto t1=new Toto();
Toto t2=new Toto();
```

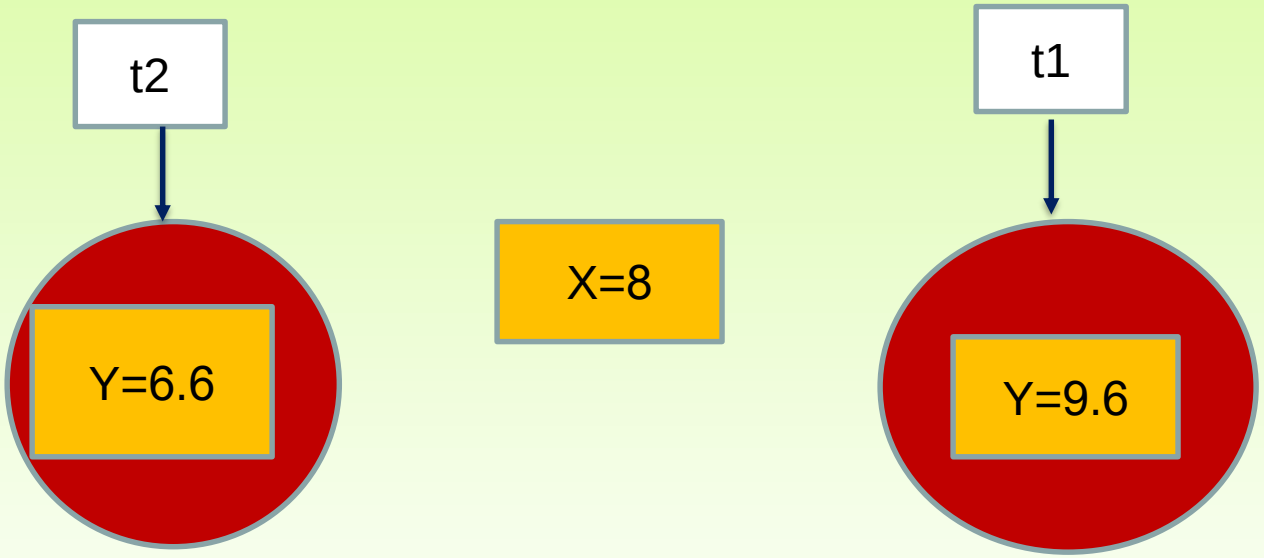
يمكننا تمثيل هذه الأغراض بالشكل التالي:



إلى هنا لم نلاحظ أي فرق لنكتب بالتعليمات التالية:

t2.y=9.6; t1.x=8; t1.y=6.6;

هنا سنجد أن التمثيل البياني أصبح بالشكل:



ملاحظات حول الطرق Methods

- يعرف كل صف مجموعة من الطرق التي تقوم بتنفيذ مهام محددة على حقول الصف.
- بعد تعريف الطريقة في صف معين، يمكن استدعاؤها من قبل أي طريقة أخرى بهدف الحصول على الخدمة التي توفرها هذه الطريقة
- الشكل العام للطريقة ضمن الصف

```
Access -modifier return Type method Name(parameters list){
    Method Body
}
```

- محدد الوصولAccess-modifier: يحدد سماحية الوصول إلى الطريقة، أي طريقة الوصول للطريقة.
- Return Type:يحدد النوع الذي سترجعه الدالة عندما تنتهي ويمكن إن يكون voidفلا ترجع الطريقة أي قيمة
- Method Name:يمثل الاسم الذي تعطيه للدالة والذي من خلالها يمكننا استدعائها
- Parameters list:قائمة الوسطاء
- Method Body: قد يحوي جسم الدالة على متغيرات محلية Local Variable تستخدم للتخزين المؤقت للبيانات ضمن الطريقة، كما يحوي مجموعة تعليمات برمجية تقوم بتنفيذ العمل المحدد على الطريقة.

Method Header(Signature) ترويسة الطريقة

- A method declaration begins with a *method header*

```
char calc (int num1, int num2, String message)
```

return type

method name

parameter list

The parameter list specifies the type and name of each parameter

تحدد قائمة الوسطاء نوع و اسم كل وسيط

Method Body جسم الطريقة

- The method header is followed by the *method body*

```
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```

متحولات محلية
يتم انشاؤهم كل مرة تستدعى فيها الطريقة، و يتم التخلص منهم بعد انتهاء تنفيذ الطريقة

The return expression must be consistent with the return type

`sum` and `result` are local data

They are created each time the method is called, and are destroyed when it finishes executing

Parameters الوسطاء

```
ch = obj.calc (25, count, "Hello");
```

```
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```

التحميل الزائد للطرق
Methods Overloading

- هو إمكانية كتابة أكثر من طريقة في نفس الصف تحمل نفس الاسم على أن نغير في نوع الوسطاء أو عددها أو ترتيبها إذا كانت من أنواع مختلفة.
- لا يكون هناك تحميل زائد للطرق إذا كانت الطرق مختلفة بنوع الارجاع.

Method Overloading التحميل الزائد للطريقة

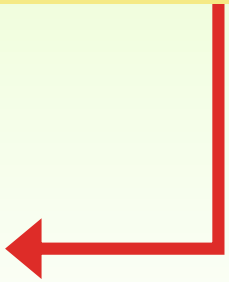
- The compiler determines which method is being invoked by analyzing the parameters

```
float tryMe(int x)
{
   return x + .375;
}

float tryMe(int x, float y)
{
   return x*y;
}
```

Invocation (الاستدعاء)

```
result = tryMe(25, 4.32)
```



Method Overloading

- The `println` method is overloaded:

```
println (String s)
println (int i)
println (double d)
```

and so on...

- The following lines invoke different versions of the `println` method:

```
System.out.println ("The total is:");
System.out.println (total);
```

أمثلة: هل الحالات التالية تحميل زائد؟ولماذا؟

char F(int x,int y);	**الطريقة الأولى وهي الأساسية**
char F(int y,int x);	خطأ، لان نوع وسطائها نفس الاساسية
int F(int y);	صحيحة، لاختلاف عدد الوسطاء
int F(double x,int y);	صحيحة، لاختلاف انواع الوسطاء
int F (int x,int y);	خطأ، لأنها مطابقة للأساسية ونوع الارجاع لا يهم

مثال :

```
Int  F(int x,double y, char z);

Int  F(double y,int x,char z);

Int  F(char z,double y,int x);
```

تحميل زائد مقبول لوجود اختلاف بترتيب أنواع الوسطاء

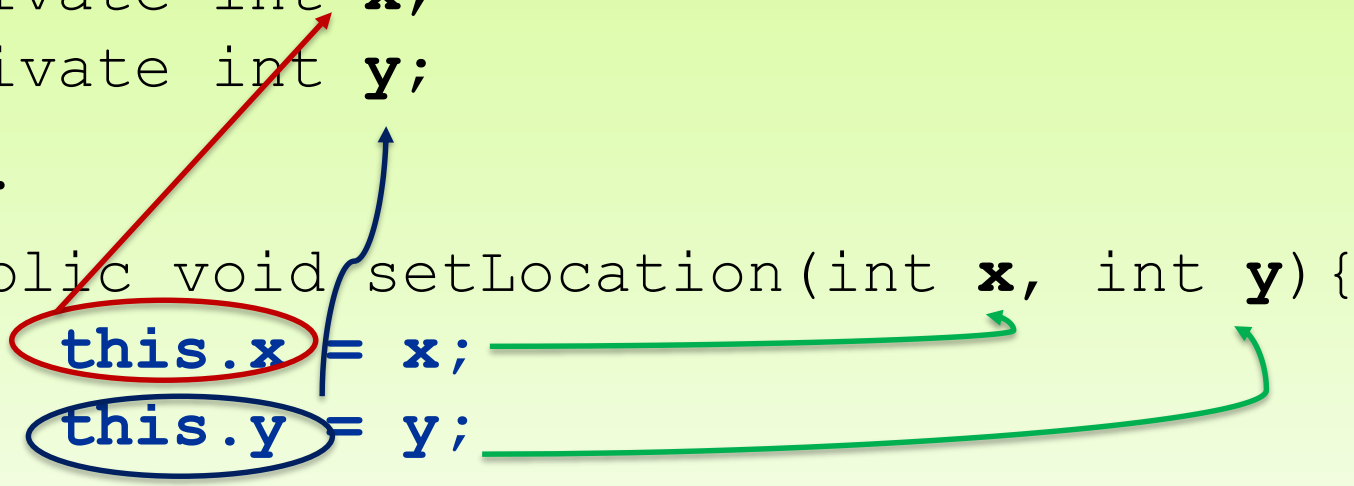
ملاحظات حول الكلمة المفتاحية this

The this Keyword

- The this keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class's *hidden data fields*.
  - الكلمة المفتاحية this تستخدم للإشارة إلى الغرض الحالي من صف معين
  - للتمييز في جسم الطريقة بين وسيط الطريقة و الحقل فيجب عندئذ استخدام الكلمة المفتاحية this قبل اسم الحقل في جسم الطريقة.
- Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.
  - استخدام آخر للكلمة المفتاحية this هو تمكين باني من استدعاء باني آخر من نفس الصف. عند استدعاء الباني بوساطة this فإن هذا الاستدعاء يجب أن يكون أول تعليمة في الباني الذي يقوم بالاستدعاء.

Example

```
public class Point {
    private int x;
    private int y;
    ...
    public void setLocation(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```



- Inside `setLocation`,
  - To refer to the data field `x`, say `this.x`
  - To refer to the parameter `x`, say `x`

Example

```
public class Point {
  private int x;
  private int y;
  private int i, k;
  public Point() {
    this(0, 0);// calls (x, y) constructor
  }
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public Point(int x, int y, int i, int k){
    this(x,y);
    //يجب التأكد من تطابق عدد الوسطاء و نوعها في التعليمة this مع وسطاء الباني المطلوب استدعاؤه
    this.i = i;
    this.k=k;
  }


}
```

Example

```
public class F {
  private int i = 5;
  private static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    F.k = k;
  }
}
```

للتمييز في جسم الطريقة بين وسيط الطريقة i و الحقل i فيجب عندئذ استخدام الكلمة المفتاحية this قبل اسم الحقل في جسم الطريقة.

Example

```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }
}
```

this must be explicitly used to reference the data field radius of the object being constructed

this is used to invoke another constructor

Every instance variable belongs to an instance represented by this, which is normally omitted

البواني Constructors

- الباني طريقة خاصة لها اسم و يمكن أن تقبل مجموعة من الوسطاء
- يجب أن يطابق اسم الباني اسم الصف المعرف فيه
- يختلف الباني عن الطرق العادية في أنه لا يملك نوع ارجاع لأنه لا يعيد ظاهريا أي قيمة لكنه يعيد ضمنيا غرض من نوع الصف
- يمكن أن يحتوي الصف على أي عدد من البواني شرط أن تكون مختلفة عن بعضها البعض بعدد الوسطاء أو نوعها
- هناك ثلاثة بواني في الصف Person. الاختلاف بين الأول و الثاني بنوع الوسيط. الوسيط في الباني الأول من النوع الصحيح و في الباني الثاني من النوع الحقيقي. الباني الثالث مختلف بنوع و عدد الوسطاء. يستطيع مترجم الجافا معرفة أي بان يستخدم من خلال نوع القيمة

البواني Constructors

تستخدم البواني للقيام بالعديد من المهام نذكر منها:

- انشاء أغراض objects من صف معين. يتم ذلك باستخدام الكلمة المفتاحية new. أي أن الباني يتم استخدامه لمرة واحدة في اللحظة التي يتم فيها انشاء غرض من الصف.
- يقوم الباني إعطاء قيم ابتدائية لحقول الغرض المنشأ من صف معين.
- لكل صف دوما باني يجب تعريفه، و في حال عدم تعريف أي باني للصف فإن المترجم يقوم ضمنيا بإنشاء باني افتراضي للصف (وهو باني لا يحوي وسطاء ولا يحوي أية تعليمات) .
- لا تشترط لغة الجافا كتابة باني بشكل صريح في الصف. فلغة الجافا تعرف الباني الافتراضي (باني بدون وسطاء) إذا لم يكتب المبرمج باني فعلي لإنشاء الأغراض
- عند كتابة أي باني صراحة (بوسطاء أو بدون وسطاء) يقوم المترجم بإلغاء الباني الافتراضي ولا يمكن استدعائه
- إذا أردنا استخدام باني افتراضي في صف معرف فيه باني غير افتراضي فيجب تعريف الباني الافتراضي بشكل صريح

Defining and Overloading Constructors

تقنية الـ Constructor Overloading في الجافا تمكن المبرمج تعريف أكثر من باني Constructor، يختلف أحدها عن الآخر بقائمة الوسطاء Parameter List

يتم التفريق بين البواني بأخذ عدد الوسطاء، ونوعها بعين الاعتبار.

```
public class Movie {
private String title;
private String rating="PG";
public Movie(){
   title="Last Action";
}
public Movie(String newTitle){
   title=newTitle;
}
```

```
Movie mov1 = new Movie();
Movie mov2 = new Movie("Inception");
```

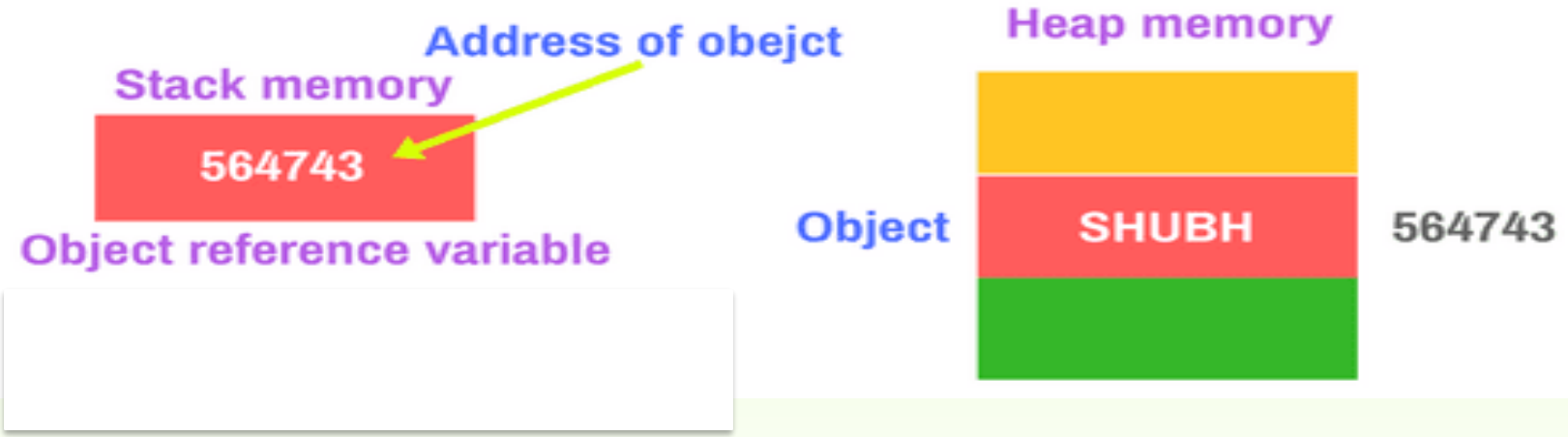
How to create an object of the class?

Keyword that stores the object in heap memory

Class_name Object_reference_variable_name=new Class_name()

stores address of the object on the stack memory

Constructor



Declaring Object Reference Variables
التصريح عن متغيرات مرجعية لغرض

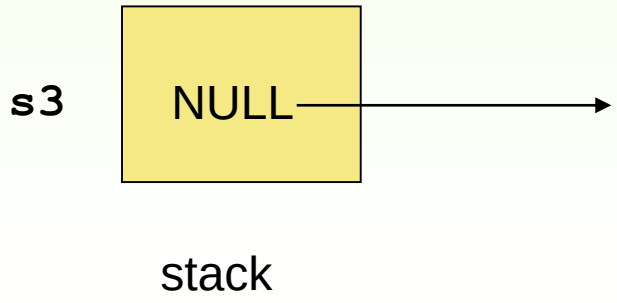
```
ClassName objectReference;
```

Example:

للتصريح عن المتغير المرجعي s3 و يشير إلى عنوان غرض من الصف، نكتب:

```
Student s3;
```

يتم حجز مكان في الذاكرة stack اسمه s3 فارغ (أي لا يشير إلى أي غرض أي بمعنى أنه لا يحمل عنوان لأي غرض بعد)



Creating Objects انشاء الأغراض

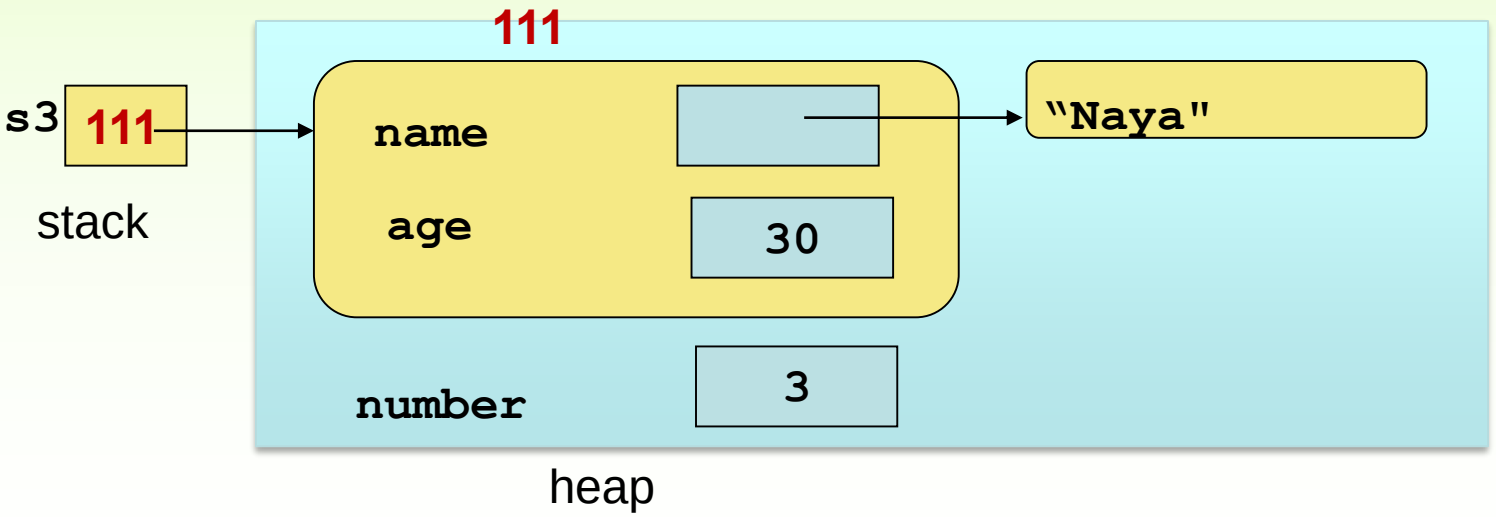
```
objectReference = new ClassName();
```

Example:

```
s3 = new Student("Naya", 30);
```

The object reference is assigned to the object reference variable.

يتم حجز مكان بالذاكرة Heap (له عنوان وليكن 111) للغرض المنشأ من الصف، و من ثم يتم جعل s3 يشير إلى الغرض المنشأ (أي يحمل عنوان الغرض المنشأ في الذاكرة Heap)

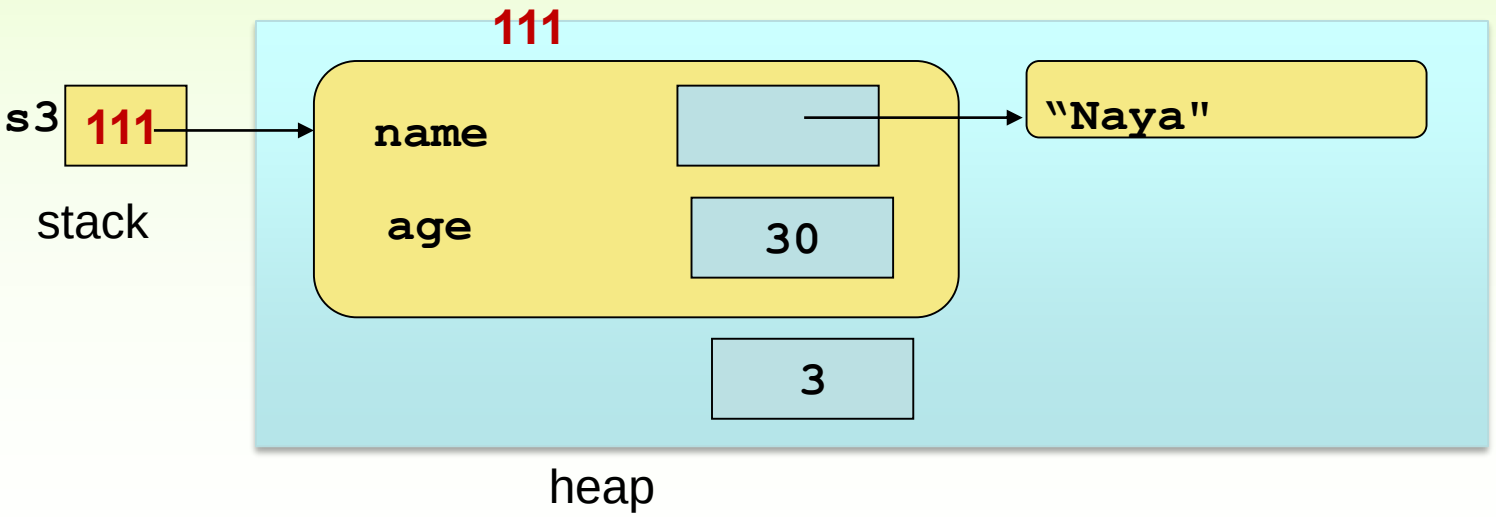


Declaring/Creating Objects in a Single Step

```
ClassName objectReference = new ClassName();
```

Example:

```
Student s3 = new Student("Naya", 30);
```

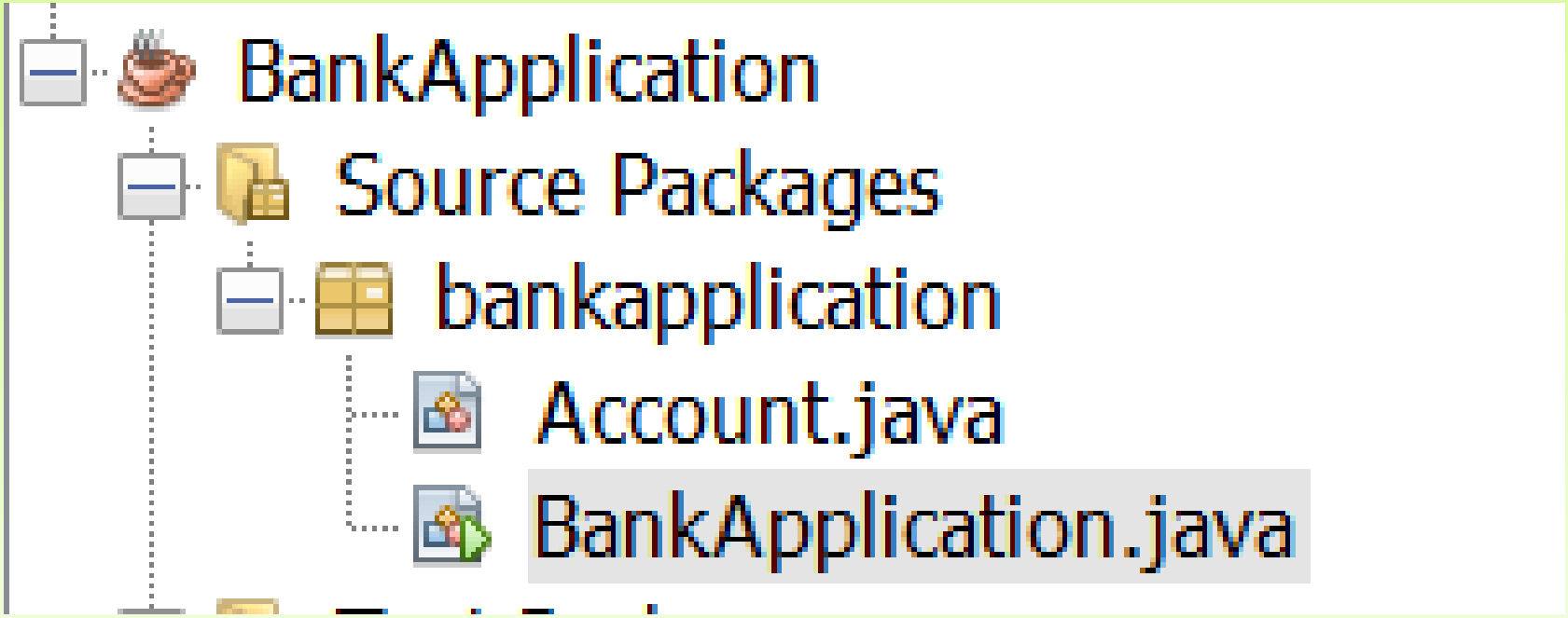


اكتب برنامجا بلغة الجافا يقوم بما يلي:

- **تعريف صف Account يحتوي المعلومات التالية:**
  - **الحقول:** رقم الحساب accountNum، اسم مالك الحساب cname، قيمة الرصيد balance، و الرقم الوطني لمالك الحساب cid
  - **الطرق:**
    - باني يستقبل الحقول السابقة
    - طرق لتعديل الحقول السابقة
    - طرق للحصول على الحقول السابقة
    - طريقة لإضافة مبلغ للحساب
    - طريقة لسحب مبلغ من الحساب
    - طريقة لنقل مبلغ من الحساب الحالي إلى حساب آخر
    - طريقة لطباعة معلومات الحساب
- **تعريف صف رئيسي BankApllication يقوم بما يلي:**
  - ❖ إنشاء حسابين
  - ❖ طباعة معلومات الحسابين
  - ❖ نقل مبلغ 1000 من الحساب الأول إلى الحساب الثاني

تمثيل الصف Accountباستخدام لغة UML

Account
- accountNum:String - cname: String - balance:double - cid:String
+Student(accountNum:String, cname:String, balance:double, cid:String) +set_accountNum (accountNum:String):void +set_cname(cname:String):void +set_balance(balance:double):void +set_cid(cid:String):void +get_accountNum():String +get_cname():String +get_balance():double +get_cid():String +deposite(amount:double):void +withdrow(amount:double):void + transact(a:Account, amount:double):void +showAccountInfo(): void


BankApplication
Source Packages
bankapplication
Account.java
BankApplication.java

تعريف حقول الصف Account و الباني

```
package bankapplication;
public class Account {
  private String accountNum;
  private String cname;
  private double balance;
  private String cid;
  public Account(String accountNum,String cname,double balance,String cid) {
      this.accountNum=accountNum;
      this.cname=cname;
      this.balance=balance;
      this.cid=cid;
  }
```

تعريف طرق الـ set لحقول الصف Account

```
public void set_accountnum(String accountNum) {
    this.accountNum=accountNum;
}
public void set_cname(String cname)  {
    this.cname=cname;
}
public void set_balance(double balance) {
    this.balance=balance;
}
public void set_cid(String cid) {
    this.cid=cid;
}
```

تعريف طرق الـ get لحقول الصف Account

```
public String get_accountNum(){
    return accountNum;
}
public String get_cname()  {
    return cname;
}
public double get_balance(){
    return balance;
}
public String get_cid(){
    return cid;
}
```

تعريف طرق السحب و الإيداع و التحويل

```
public void deposite(double ammount){
    balance=balance+ammount;
}
public void withdrow(double ammount){
    if (ammount<balance)
        balance=balance-ammount;
    else
        System.out.println("you don't have this ammount to withdrow");
}
public void transact(Account a,double ammount){
    if(ammount<balance)
      {
         balance=balance-ammount;
         a.deposite(ammount);
      }
    else System.out.println("you can't do this transaction!");
 }
```

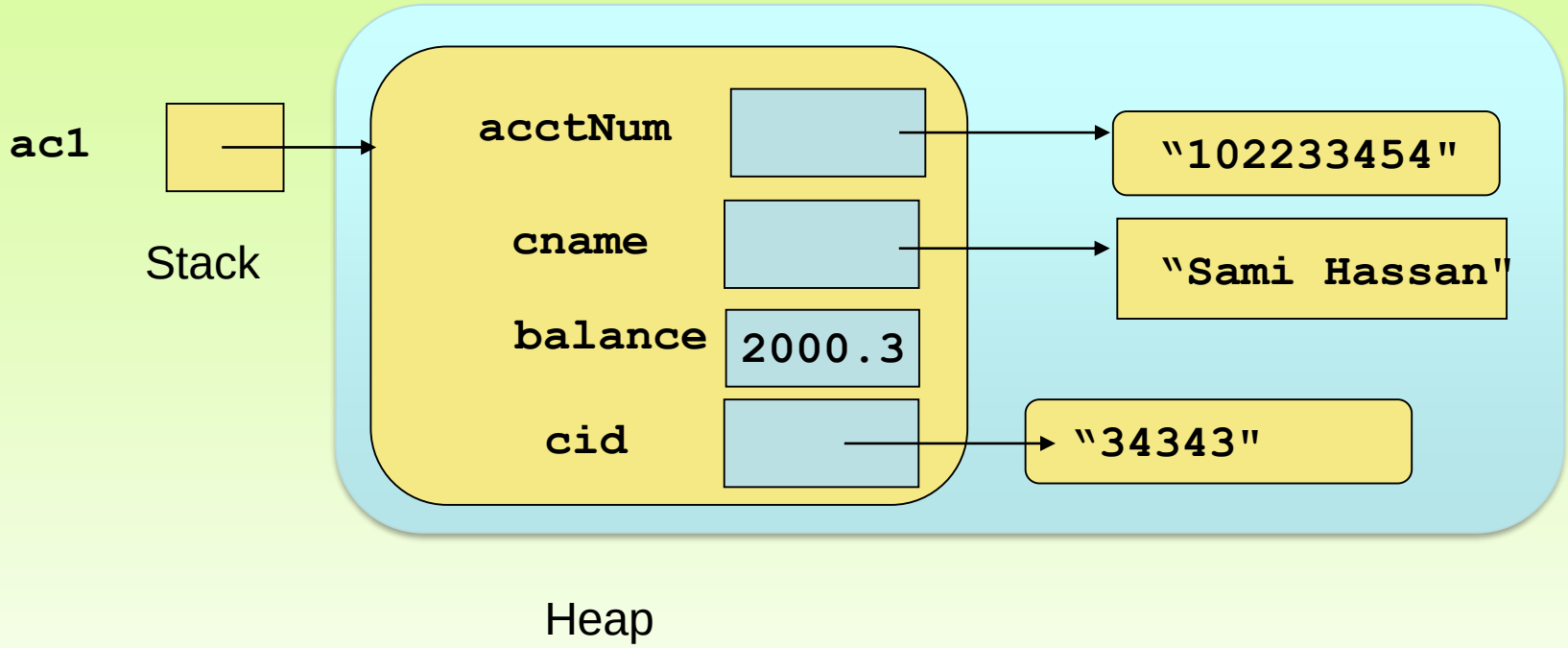
تعريف طريقة طباعة معلومات كل حساب

```
    public void showAccountInfo(){
        System.out.println("hello "+cname+" your account number is: "+accountNum+
                "  your ID is" +cid+" your balance is:"+balance);
    }
}
```

BankApplication الصف

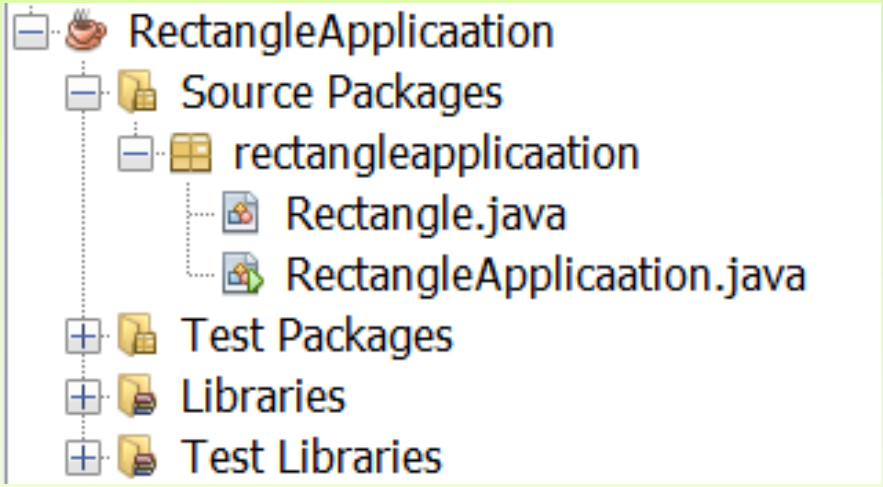
```
package bankapplication;
public class BankApplication {
    public static void main(String[] args) {
        Account ac1=new Account("102233454","Sami Hassan",2000.3,"33454");
        ac1.set_balance(3000);
        Account ac2=new Account ("102234343","Maya Ali",8000,"34343");
        ac1.showAccountInfo();
        ac2.showAccountInfo();
        ac1.transact(ac2, 1000);
        ac1.showAccountInfo();
        ac2.showAccountInfo();
    }
}
```

Bank Account Example



اكتب برنامجا بلغة الجافا يقوم بما يلي:

- **تعريف صف Rectangle يحتوي المعلومات التالية:**
  - **الحقول:** طول المستطيل length، عرض المستطيل width
  - **الطرق:**
    - باني يستقبل الحقول السابقة
    - باني بدون وسطاء
    - طرق لتعديل الحقول السابقة
    - طرق للحصول على الحقول السابقة
    - طريقة لحساب المساحة
    - طريقة لحساب المحيط
    - طريقة لجمع مساحة المستطيل الحالي إلى مساحة مستطيل آخر
    - طباعة معلومات المستطيل
- **تعريف صف رئيسي RectangleApllication يقوم بما يلي:**
  - ❖ إنشاء مستطيلين
  - ❖ طباعة معلومات عن المستطيلين
  - ❖ جمع مساحة المستطيل الثاني إلى الأول


RectangleApplicaation
Source Packages
rectangleapplicaation
Rectangle.java
RectangleApplicaation.java
Test Packages
Libraries
Test Libraries

تعريف حقول الصف Rectangle و البواني

```
package rectangleapplicaation;
public class Rectangle {
   private double length;
   private double width;
   Rectangle(double length,double width)   {
      this.length=length;
      this.width=width;
   }
   Rectangle() {
      length=5;
      width=6;
   }
```

تعريف طرق الـ set و الـ get لحقول الصف Rectangle

```
public double getwidth()   {
   return width;
}
public double getlength(){
   return length;
}
public void setwidth(double width){
   this.width=width;
}
public void setlength(double length){
   this.length=length;
}
```

تعريف طرق حساب المساحة و المحيط و جمع مساحة مستطيل إلى المستطيل الذي يتم التعامل معه حاليا، إضافة لطريقة طباعة معلومات عن كل مستطيل منشأ

```
  public double area(){
    return length*width;
  }
   public double perimeter(){
    return 2*(length+width);
  }
  public double sum_area(Rectangle a) {
    return (area()+a.area());
  }
  public void printinfo()  {
    System.out.println("width is: "+width);
    System.out.println("length is:"+length);
    System.out.println("area is:"+area());
    System.out.println("perimeter is:"+perimeter());
    System.out.println("**********************");
  }

}
```

RectangleApplication الصف

```
package rectangleapplicaation;
public class RectangleApplicaation {
    public static void main(String[] args) {
      Rectangle a=new Rectangle();
      Rectangle b=new Rectangle(6.7,8.9);
      System.out.println("a rectangle: ");
      a.printinfo();
      System.out.println("********************");
      System.out.println("b rectangle: ");
      b.printinfo();
      System.out.println("sum of a area and b area is : "+a.sum_area(b));
    }
}
```

التنفيذ

a rectangle:
width is: 6.0
length is:5.0
area is:30.0
perimeter is:22.0
****
****
b rectangle:
width is: 8.9
length is:6.7
area is:59.63
perimeter is:31.200000000000003
****
sum of a area and b area is : 89.63