1.什么是类和对象?
类是一组具有相同特征和功能的事物的抽象
对象描述了一个物体的特征和行为实现
类是对象的抽象 对象是类的实例
2.OC中定义类,创建对象,使用对象?
OC中定义类分为接口部分,实现部分
接口部分:对外声明类的成员变量和行为
实现部分:实现方法,即类的行为实现
创建对象需要进行分配内存空间和初始化
3.继承.继承的所有特点?
子类可以继承父类所有的声明的方法和实例变量,私有的成员变量也继承了但是不能直接访问
父类不能使用子类所创建的方法和实例变量 (强转为子类类型后可以)
4.self和super?
self 给对象\类发送消息 执行该对象\类的方法
super 给对象\类发送消息 执行父类的对象\类方法
5.初始化方法,根据需求自定义,能够写出完整的实现?
例: - (id)initWithName:(NSString *)name Sex:(NSString *)sex
Age:(int)age
{

```
if(self = [super init])
{
  self.name = name;
  self.sex = sex;
  _age = age
}
return self;
```

}
6.实例变量的可见度,以及各自的特点,默认可见度是什么?
如何在类的外部,操作默认可见度的实例变量.
@public 公有的
@protected 保护的(默认的)
@private 私有的
@package(框架级别)作用域介于私有和公开之间,只要处于同一个框架中就可以直接通过变量名访问
在类的外部,访问默认可见度的实例变量 可以定义实例变量的属性,也可以声明和实现setter和getter方法
7.什么是setter,getter方法,如何声明以及实现?

Setter： - (void) setName:(NSString *)name;
Getter： - (NSString *) name;

8.属性的作用.如何声明属性,如何实现属性(声明和实现所对应的关键字)？
作用:提供外部访问 自动生成setter和getter方法声明和实现
@property (属性的属性) 返回值或参数类型 属性的名字
@synthesize 属性的名字 = 实例变量 (可省略)

9.属性的三个特性?特性都有哪些内容,使用场景,特点作用？
读写设置:readonly(只读) readwrite(默认)
语义设置: assign(默认 既可以修饰基本数据类型还可以修饰对象)
retain(修饰对象)
copy(修饰对象 并且所修饰的对象要遵守<NSCopying>协议)
多线程:nonatomic(不能保证多线程安全 性能高)
atomic(默认 保证多线程安全 性能低)

10.对象属性的setter和getter方法的内部实现？
用retain修饰

```
- (void)setName:(NSString *)name
{
    if(_name != name)
    {
        [_name release];
        _name = [name retain];
    }
}
- (NSString *)name
{
    return [[_name retain] autorelease];
}
```

11.类的扩展.(分类 延展 协议(一对方法)).代理(执行协议里方法的对象)？
分类(category):为(没有源代码的)类添加方法 没有实例变量
分类的声明: @interface 类名 (分类名称)
@end
分类的实现: @implementation 类名
@end
注:分类的方法若与类的方法相同 分类的方法优先级高 (覆盖)
延展(extention):(匿名分类)管理"私有"方法 写在.m文件中 可以有实例变量
延展的声明:@interface 类名 ()
@end

注:声明在延展里的方法必须实现
协议(protocol):没有方法实现 只有方法声明 只有.h文件
关键字@required 必须实现 (默认, 不实现只是警告)
@optionol 可以不实现
协议的声明:@protocol 协议的名字 <NSobject>
注:一个类可以接受多个协议,在< >中用","分隔开.用协议可以实现多继承

12.内存管理?
对内存管理的理解
对象所有权:通过retain alloc copy 使对象的引用计数器加1
内存泄露:只分配使用了一块内存,没有释放
过度释放:释放多次
野指针:指针指向不存在的对象(不可用内存)

13.怎么确定一个方法的方法名?
去掉减号,加号,去掉返回值,去掉参数类型,去掉参数,剩下的就是方法名了

14.字典集合的特点?
字典是以键值对形式的保存元素 key-->Value
只能存储对象,不能存储基本数据类型

15.动态绑定?
NSString * testObject = [[NSData alloc]init]
编译时是NSString类型 (不检测)
运行时是NSData类型 (检测)

16.分类,延展,协议,继承的区别?
分类:分类可以在不获悉,不改变原来代码的情况下往里面添加新的方法,只能添加,不能删除原有方法,但是无法添加实例变量
延展:延展可以同时添加实例变量和方法,而且添加的方法必须实现,可以视为一个私有的分类
协议:协议是多个类共享的一个方法列表,在协议中只有方法的声明
继承:不但可以添加实例变量和方法,也可以重写原有类的方法

17.变量类型?
<1>局部变量和全局变量:①局部变量:在函数代码块内部定义的变量,作用域为只能在定义该局部变量的函数内部使用,从定义变量的那一行开始,一直到代码块结束;②全局变量:在所有函数外部定义的变量, 作用域为从定义变量的位置开始到源程序结束, 即全局变量可以被在其定义位置之后的其他函数所共享.

<2>变量的存储类型是指变量存储在什么地方,有3个地方可以用于存储变量:普通内存,运行时堆栈,硬件寄存器:①自动变量:自动变量是存储在堆

栈中的,被关键字auto修饰的局部变量都是自动变量,所有的局部变量在默认情况下都是自动变量;生命周期:在程序执行到声明自动变量的代码块(函数)时,自动变量才被创建;当自动变量所在的代码块(函数)执行完毕后,这些自动变量就会自行销毁.如果一个函数被重复调用,这些自动变量每次都会重新创建;②静态变量:静态变量是存储在静态内存中,不属于堆栈;所有的全局变量都是静态变量,被关键字static修饰的局部变量也是静态变量;生命周期:静态变量在程序运行之前创建,在程序的整个运行期间始终存在,直到程序结束;③寄存器变量:存储在硬件寄存器中的变量,寄存器变量比存储在内存中的变量访问效率更高(默认情况下,自动变量和静态变量都是放在内存中的);被关键字register修饰的自动变量都是寄存器变量,只有自动变量才可以是寄存器变量,全局变量和静态变量都不行,寄存器变量只限于int,char和指针类型变量使用;生命周期:因为寄存器变量本身就是自动变量,所以函数中的寄存器变量在调用该函数时占用寄存器中存放的值,当函数结束时释放寄存器,变量消失.

18.指针变量？

<1>指针变量的定义: 类名标识符 *指针变量名;

<2>64位系统环境下,所有的指针都是8个字节,指针是用来装地址的,且只能存储地址;

<3>地址是常量,指针是变量(只有变量才有地址),指针是地址变量,数组名是地址常量;

<4>指针就一个作用:能够根据一个地址值,访问对应的存储空间;

<5>指针变量未经初始化,不能拿来间接访问其他存储空间.

19.预处理指令？

<1>宏定义,有不带参数的宏定义,带参数的宏定义,终止宏定义的作用域,可以用#undef命令;宏定义不涉及存储空间的分配,参数类型分配,参数传递,返回值问题,函数调用在程序运行时执行,而宏替换只在编译预处理阶段进行,所以带参数的宏比函数具有更高的执行效率;

<2>条件编译:希望程序的其中一部分代码只有在满足一定条件时才进行编译,否则不参与编译(只有参与编译的代码最终才能被执行);

<3>文件包含:#include,它可以将一个文件的全部内容拷贝到另一个文件中;①#include <文件名>②#include “文件名”

17.NSString 不可变字符串的常用方法？

<1>@property (readonly) NSUInteger length; 只读返回字符串中的字符个数;

<2>- (unichar)characterAtIndex:(NSUInteger)index; 取得字符串索引处的字符;

<3>- (NSString *)substringFromIndex:(NSUInteger)from; 返回一个字符串,截取当前字符串对象范围是给定索引到这个字符串的结尾;

<4>- (NSString *)substringToIndex:(NSUInteger)to; 返回一个字符串，截取当前字符串对象范围是从索引 0 到给定的索引;

<5>- (NSComparisonResult)compare:(NSString *)string; 字符串与 string 字符串逐个字符比较;

<6>- (BOOL)isEqualToString:(NSString *)aString; 返回 BOOL 值的字符串比较函数,比较两个字符串的内容是否相等;

<7>- (BOOL)hasPrefix:(NSString *)aString; 检查字符串是否以另一个字符串开头;

<8>- (BOOL)hasSuffix:(NSString *)aString; 检查字符串是否以另一个字符串结尾;

<9>- (NSRange)rangeOfString:(NSString *)aString; 返回字符串中是否包含 aString 字符串的范围;

<10>- (NSString *)stringByAppendingString:(NSString *)aString; 在当前字符串后拼接一个字符串,返回新的字符串;

<11>@property (readonly) int intValue; 将字符串转为 int 类型;

<12>@property (readonly, copy) NSString *uppercaseString; 将字符串全部转为大写,返回新的字符串;

<13>@property (readonly, copy) NSString *lowercaseString; 将字符串全部转为小写,返回新的字符串;

<14>@property (readonly, copy) NSString *capitalizedString; 将字符串首字母大写,其他字母都变小写,返回新的字符串;

<15>- (NSArray*)componentsSeparatedByString:(NSString *)separator; 用 separator 为分隔符截取子串，返回一个装着所有子串的集合 NSArray;

18.NSMutableString 可变字符串的常用方法？

<1>- (void)insertString:(NSString *)aString atIndex:(NSUInteger)loc; 将 aString 插入到当前字符串的 loc 索引处;

<2>- (void)deleteCharactersInRange:(NSRange)range; 删除当前字符串 range 范围内的字符;

<3>- (void)appendString:(NSString *)aString; 将字符串 aString 添加到当前字符串末尾;

<4>- (void)setString:(NSString *)aString; 初始化完毕后可以使用此方法设置字符串的内容;

<5> - (void)replaceCharactersInRange:(NSRange)range withString:(NSString *)aString; 将 range 位置的字符串替换为 aString;

19.NSArray 不可变数组的常用方法？

<1>@property (readonly) NSUInteger count; 返回数组中的对象个数;

<2>- (id)objectAtIndex:(NSUInteger)index; 获得数组中索引处的对象;

<3>- (NSArray *)arrayByAddingObject:(id)anObject; 将 anObject 对象添加到数组中,返回一个新的数组;

<4>- (NSString *)componentsJoinedByString:(NSString *)separator; 合并数组中的元素并创建字符串;

<5>- (NSUInteger)indexOfObject:(id)anObject; 查找 anObject 对象在数组中的位置;

<6>- (NSUInteger)indexOfObject:(id)anObject inRange:(NSRange)range; 在 range 范围内查找元素的位置;

<7>- (BOOL)isEqualToArray:(NSArray *)otherArray; 比较两个数组内容是否相同;

<8>- (id)firstObjectCommonWithArray:(NSArray *)otherArray; 返回两个数组中第一个相同的对象元素;

<9>- (void)makeObjectsPerformSelector:(SEL)aSelector; 让集合里面的所有元素都执行 aSelector 这个方法;

<10>- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)otherArray; 添加 otherArray 数组的所有元素,返回一个新的 NSArray(方法调用者本身没有改变);

<11>- (NSArray *)subarrayWithRange:(NSRange)range; 截取 range 范围的数组元素;

20.NSArray 遍历?

<1>普通遍历:

```
NSUInteger count = [array count];
for (int i = 0; i < count; i++)
{
    id obj = [array objectAtIndex:i];
}
```

<2>快速遍历:

```
for (id obj in array)
```

<3>迭代器遍历(查看 NSEnumerator 的用法)

<4> block 遍历:

```
[array enumerateObjectsUsingBlock:^(id object, NSUInteger index,
BOOL *stop) {
    NSLog(@"%@ - %zi", object, index);
}];
```

21.NSEnumerator 集合的迭代器,可用于遍历集合元素?

<1>- (NSEnumerator *)objectEnumerator; 获取一个正序遍历的迭代器;

<2>- (NSEnumerator *)reverseObjectEnumerator; 获取一个反序遍历的迭代器;

<3>- (id)nextObject; 获取下一个元素;
<4>- (NSArray *)allObjects; 获取没有被遍历过的元素;
22.NSArray 排序？
<1>- (NSArray *)sortedArrayUsingDescriptors:(NSArray *)sortDescriptors; 这个方法接收一组 NSSortDescriptor 对象作为参数,每个 NSSortDescriptor 对象指定一个 key 路径和排序方向(升序或者降序),数组中 NSSortDescriptors 的顺序决定每个字段的优先级,key 路径允许用字段名称来 get\set 对象字段,访问子孙对象的字段,需要使用”.”这个分隔符;
<2> NSSortDecriptor 的初始化方法:
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending; 默认是用 compare:作为比较方法;
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending selector:(SEL)selector; 可以传递一个 selector 设置比较方法;
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending comparator:(NSComparator)cmptr; 可以传递一个 block 来专门比较;
<3>举例:
NSSortDescriptor *desc1 = [NSSortDescriptor sortDescriptorWithKey:@“age” ascending:NO]; // 先按照 age 属性降序;
NSSortDescriptor *desc2 = [NSSortDescriptor sortDescriptorWithKey:@“name” ascending:YES]; // 再按照 name 属性升序;
NSArray *descArray = [NSArray arrayWithObjects:desc1, desc2, nil]; // 将 desc1 和 desc2 排序依据放入 descArray 数组中;
NSArray *array1 = [array sortedArrayUsingDescriptors:descArray]; // 数组 array 依据 descArray 数组排序后的生成数组 array1;
23.NSMutableArray 可变数组的常用方法？
<1>- (void)addObject:(id)anObject; 添加一个对象元素;
<2>- (void)insertObject:(id)anObject atIndex:(NSUInteger)index; 在 index 位置插入一个元素;
<3>- (void)addObjectsFromArray:(NSArray *)otherArray; 添加 otherArray 的全部元素到集合中;
<4>- (void)insertObjects:(NSArray *)objects atIndexes:(NSIndexSet *)indexes; 在 indexes 指定的位置分别插入 objects 中的元素;
<5>- (void)removeLastObject; 删除最后一个元素;
<6>- (void)removeAllObjects; 删除所有的元素;
<7>- (void)removeObjectAtIndex:(NSUInteger)index; 删除 index 位置的元素;

<8>- (void)removeObjectsAtIndexes:(NSIndexSet *)indexes;　删除 indexes 位置的所有元素;

<9> - (void)removeObject:(id)anObject;　删除特定的元素;

<10>- (void)removeObject:(id)anObject inRange:(NSRange)range; 在 range 范围内查找特定的元素进行删除;

<11> - (void)removeObjectsInArray:(NSArray *)otherArray;　删除同时存在于 otherArray 和当前集合中的所有元素;

<12>- (void)removeObjectsInRange:(NSRange)range;　删除 range 范围内的所有元素;

<13>- (void)replaceObjectAtIndex:(NSUInteger)index withObject:(id)anObject;　用 anObject 替换 index 位置对应的元素;

<14>- (void)replaceObjectsAtIndexes:(NSIndexSet *)indexes withObjects:(NSArray *)objects;　用 objects 中的元素分别替换 indexes 对应位置的元素;

<15>- (void)replaceObjectsInRange:(NSRange)range withObjectsFromArray:(NSArray *)otherArray range:(NSRange)otherRange;　用 otherArray 中 otherRange 范围内的元素替换当前集合 range 范围内的元素;

<16>- (void)replaceObjectsInRange:(NSRange)range withObjectsFromArray:(NSArray *)otherArray;　用 otherArray 中的元素替换当前集合 range 范围内的元素

<17>- (void)exchangeObjectAtIndex:(NSUInteger)idx1 withObjectAtIndex:(NSUInteger)idx2;　交换 idx1 和 idx2 位置的元素;

24.NSDictionary 不可变字典的常用方法？

<1>@property (readonly) NSUInteger count;　返回字典的 key 数;

<2>@property (readonly, copy) NSArray *allKeys;　返回所有的 key;

<3>- (NSArray *)allKeysForObject:(id)anObject;　返回 anObject 元素对应的所有 key;

<4>@property (readonly, copy) NSArray *allValues;　返回所有的 value;

<5>- (id)objectForKey:(id)aKey;　根据 aKey 返回对应的 value;

<6>- (NSArray *)objectsForKeys:(NSArray *)keys notFoundMarker:(id)marker;　返回 keys 对应的所有 value,如果没有对应的 value,用 marker 代替;

25.NSMutableDictionary 可变字典的常用方法？

<1>- (void)setDictionary:(NSDictionary *)otherDictionary;　传入一个字典设置当前的字典;

<2>- (void)setObject:(id)anObject forKey:(id <NSCopying>)aKey; 添加

一个键值对;

<3>- (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary; 添加 otherDictionary 的所有元素到当前字典中;

<4>- (void)removeAllObjects; 删除所有元素;

<5>- (void)removeObjectForKey:(id)aKey; 删除 aKey 对应的值;

<6>- (void)removeObjectsForKeys:(NSArray *)keyArray; 删除 keyArray 中所有 key 对应的值.