```
class Adafruit_ILI9341 : public Adafruit_SPITFT {
public:
  Adafruit_ILI9341(int8_t _CS, int8_t _DC, int8_t _MOSI, int8_t _SCLK, int8_t _RST = -1, int8_t _MISO = -1);

  void begin(uint32_t freq = 0);
  void setRotation(uint8_t r);
  void invertDisplay(bool i);
  void scrollTo(uint16_t y);
  void setScrollMargins(uint16_t top, uint16_t bottom);
  void setAddrWindow(uint16_t x, uint16_t y, uint16_t w, uint16_t h);
  uint8_t readcommand8(uint8_t reg, uint8_t index = 0);
```

```
class Adafruit_SPITFT : public Adafruit_GFX {
public:
  Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc, int8_t mosi,
                  int8_t sck, int8_t rst = -1, int8_t miso = -1);
  Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc,
                  int8_t rst = -1);
  Adafruit_SPITFT(uint16_t w, uint16_t h, tftBusWidth busWidth, int8_t d0,
                  int8_t wr, int8_t dc, int8_t cs = -1, int8_t rst = -1,
                  int8_t rd = -1);

  ~Adafruit_SPITFT(){};

  virtual void begin(uint32_t freq) = 0;
  virtual void setAddrWindow(uint16_t x, uint16_t y, uint16_t w,
                             uint16_t h) = 0;

  void initSPI(uint32_t freq = 0, uint8_t spiMode = SPI_MODE0);
  void setSPISpeed(uint32_t freq);
  void startWrite(void);
  void endWrite(void);
  void sendCommand(uint8_t commandByte, uint8_t *dataBytes,
                   uint8_t numDataBytes);
  void sendCommand(uint8_t commandByte, const uint8_t *dataBytes = NULL,
                   uint8_t numDataBytes = 0);
  void sendCommand16(uint16_t commandWord, const uint8_t *dataBytes = NULL,
                     uint8_t numDataBytes = 0);
  uint8_t readcommand8(uint8_t commandByte, uint8_t index = 0);
  uint16_t readcommand16(uint16_t addr);
  void writePixel(int16_t x, int16_t y, uint16_t color);
  void writePixels(uint16_t *colors, uint32_t len, bool block = true,
                   bool bigEndian = false);
  void writeColor(uint16_t color, uint32_t len);
  void writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                     uint16_t color);
  void writeFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  void writeFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);

  inline void writeFillRectPreclipped(int16_t x, int16_t y, int16_t w,
                                      int16_t h, uint16_t color);

  void dmaWait(void);
  bool dmaBusy(void) const; // true if DMA is used and busy, false otherwise
  void swapBytes(uint16_t *src, uint32_t len, uint16_t *dest = NULL);

  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void pushColor(uint16_t color);

  using Adafruit_GFX::drawRGBBitmap; // Check base class first
  void drawRGBBitmap(int16_t x, int16_t y, uint16_t *pcolors, int16_t w,
                     int16_t h);

  void invertDisplay(bool i);
  uint16_t color565(uint8_t r, uint8_t g, uint8_t b);

  void spiWrite(uint8_t b);           // Write single byte as DATA
  void writeCommand(uint8_t cmd);     // Write single byte as COMMAND
  uint8_t spiRead(void);              // Read single byte of data
  void write16(uint16_t w);           // Write 16-bit value as DATA
  void writeCommand16(uint16_t cmd); // Write 16-bit value as COMMAND
  uint16_t read16(void);              // Read single 16-bit value

  void SPI_WRITE16(uint16_t w); // Not inline
  void SPI_WRITE32(uint32_t l); // Not inline

  void SPI_CS_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *csPortSet = 1;
#else  // !KINETISK
    *csPortSet = csPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *csPort |= csPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_cs, HIGH);
#endif // end !USE_FAST_PINIO
  }

  void SPI_CS_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *csPortClr = 1;
#else  // !KINETISK
    *csPortClr = csPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *csPort &= csPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
```

```
#else  // !USE_FAST_PINIO
    digitalWrite(_cs, LOW);
#endif // end !USE_FAST_PINIO
  }

  void SPI_DC_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *dcPortSet = 1;
#else  // !KINETISK
    *dcPortSet = dcPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *dcPort |= dcPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_dc, HIGH);
#endif // end !USE_FAST_PINIO
  }

  void SPI_DC_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *dcPortClr = 1;
#else  // !KINETISK
    *dcPortClr = dcPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *dcPort &= dcPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_dc, LOW);
#endif // end !USE_FAST_PINIO
  }

protected:
  inline void SPI_MOSI_HIGH(void);
  inline void SPI_MOSI_LOW(void);
  inline void SPI_SCK_HIGH(void);
  inline void SPI_SCK_LOW(void);
  inline bool SPI_MISO_READ(void);
  inline void SPI_BEGIN_TRANSACTION(void);
  inline void SPI_END_TRANSACTION(void);

  inline void TFT_WR_STROBE(void); // Parallel interface write strobe
  inline void TFT_RD_HIGH(void);   // Parallel interface read high
  inline void TFT_RD_LOW(void);    // Parallel interface read low

#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
  PORTreg_t csPortSet; ///< PORT register for chip select SET
  PORTreg_t csPortClr; ///< PORT register for chip select CLEAR
  PORTreg_t dcPortSet; ///< PORT register for data/command SET
  PORTreg_t dcPortClr; ///< PORT register for data/command CLEAR
#else              // !HAS_PORT_SET_CLR
  PORTreg_t csPort;                ///< PORT register for chip select
  PORTreg_t dcPort;                ///< PORT register for data/command
#endif             // end HAS_PORT_SET_CLR
#endif             // end USE_FAST_PINIO
#if defined(__cplusplus) && (__cplusplus >= 201100)
  union {
#endif
    struct {          //   Values specific to HARDWARE SPI:
      SPIClass *_spi; ///< SPI class pointer
#if defined(SPI_HAS_TRANSACTION)
      SPISettings settings; ///< SPI transaction settings
#else
    uint32_t _freq; ///< SPI bitrate (if no SPI transactions)
#endif
      uint32_t _mode; ///< SPI data mode (transactions or no)
    } hwspi;          ///< Hardware SPI values
    struct {          //   Values specific to SOFTWARE SPI:
#if defined(USE_FAST_PINIO)
      PORTreg_t misoPort; ///< PORT (PIN) register for MISO
#if defined(HAS_PORT_SET_CLR)
      PORTreg_t mosiPortSet; ///< PORT register for MOSI SET
      PORTreg_t mosiPortClr; ///< PORT register for MOSI CLEAR
      PORTreg_t sckPortSet;  ///< PORT register for SCK SET
      PORTreg_t sckPortClr;  ///< PORT register for SCK CLEAR
#if !defined(KINETISK)
      ADAGFX_PORT_t mosiPinMask; ///< Bitmask for MOSI
      ADAGFX_PORT_t sckPinMask;  ///< Bitmask for SCK
#endif                   // end !KINETISK
#else                    // !HAS_PORT_SET_CLR
      PORTreg_t mosiPort;           ///< PORT register for MOSI
      PORTreg_t sckPort;            ///< PORT register for SCK
      ADAGFX_PORT_t mosiPinMaskSet; ///< Bitmask for MOSI SET (OR)
      ADAGFX_PORT_t mosiPinMaskClr; ///< Bitmask for MOSI CLEAR (AND)
      ADAGFX_PORT_t sckPinMaskSet;  ///< Bitmask for SCK SET (OR bitmask)
      ADAGFX_PORT_t sckPinMaskClr;  ///< Bitmask for SCK CLEAR (AND)
#endif                   // end HAS_PORT_SET_CLR
#if !defined(KINETISK)
      ADAGFX_PORT_t misoPinMask; ///< Bitmask for MISO
#endif                   // end !KINETISK
#endif                   // end USE_FAST_PINIO
      int8_t _mosi;              ///< MOSI pin #
      int8_t _miso;              ///< MISO pin #
      int8_t _sck;               ///< SCK pin #
    } swspi;                     ///< Software SPI values
    struct {                     //   Values specific to 8-bit parallel:
#if defined(USE_FAST_PINIO)

#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
      volatile uint32_t *writePort; ///< PORT register for DATA WRITE
      volatile uint32_t *readPort;  ///< PORT (PIN) register for DATA READ
#else
      volatile uint8_t *writePort;  ///< PORT register for DATA WRITE
      volatile uint8_t *readPort;   ///< PORT (PIN) register for DATA READ
#endif
#if defined(HAS_PORT_SET_CLR)
      // Port direction register pointers are always 8-bit regardless of
      // PORTreg_t -- even if 32-bit port, we modify a byte-aligned 8 bits.
#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
      volatile uint32_t *dirSet; ///< PORT byte data direction SET
      volatile uint32_t *dirClr; ///< PORT byte data direction CLEAR
```

```
#else
      volatile uint8_t *dirSet; ///< PORT byte data direction SET
      volatile uint8_t *dirClr; ///< PORT byte data direction CLEAR
#endif
      PORTreg_t wrPortSet; ///< PORT register for write strobe SET
      PORTreg_t wrPortClr; ///< PORT register for write strobe CLEAR
      PORTreg_t rdPortSet; ///< PORT register for read strobe SET
      PORTreg_t rdPortClr; ///< PORT register for read strobe CLEAR
#if !defined(KINETISK)
      ADAGFX_PORT_t wrPinMask; ///< Bitmask for write strobe
#endif                  // end !KINETISK
      ADAGFX_PORT_t rdPinMask; ///< Bitmask for read strobe
#else                   // !HAS_PORT_SET_CLR
      // Port direction register pointer is always 8-bit regardless of
      // PORTreg_t -- even if 32-bit port, we modify a byte-aligned 8 bits.
      volatile uint8_t *portDir;  ///< PORT direction register
      PORTreg_t wrPort;           ///< PORT register for write strobe
      PORTreg_t rdPort;           ///< PORT register for read strobe
      ADAGFX_PORT_t wrPinMaskSet; ///< Bitmask for write strobe SET (OR)
      ADAGFX_PORT_t wrPinMaskClr; ///< Bitmask for write strobe CLEAR (AND)
      ADAGFX_PORT_t rdPinMaskSet; ///< Bitmask for read strobe SET (OR)
      ADAGFX_PORT_t rdPinMaskClr; ///< Bitmask for read strobe CLEAR (AND)
#endif                  // end HAS_PORT_SET_CLR
#endif                  // end USE_FAST_PINIO
      int8_t _d0;               ///< Data pin 0 #
      int8_t _wr;               ///< Write strobe pin #
      int8_t _rd;               ///< Read strobe pin # (or -1)
      bool wide = 0;            ///< If true, is 16-bit interface
    } tft8;                     ///< Parallel interface settings
#if defined(__cplusplus) && (__cplusplus >= 201100)
  }; ///< Only one interface is active
#endif
#if defined(USE_SPI_DMA) &&                                          \
    (defined(__SAMD51__) ||                                          \
     defined(ARDUINO_SAMD_ZERO))     // Used by hardware SPI and tft8
  Adafruit_ZeroDMA dma;              ///< DMA instance
  DmacDescriptor *dptr = NULL;       ///< 1st descriptor
  DmacDescriptor *descriptor = NULL; ///< Allocated descriptor list
  uint16_t *pixelBuf[2];             ///< Working buffers
  uint16_t maxFillLen;               ///< Max pixels per DMA xfer
  uint16_t lastFillColor = 0;        ///< Last color used w/fill
  uint32_t lastFillLen = 0;          ///< # of pixels w/last fill
  uint8_t onePixelBuf;               ///< For hi==lo fill
#endif
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if !defined(KINETISK)
  ADAGFX_PORT_t csPinMask; ///< Bitmask for chip select
  ADAGFX_PORT_t dcPinMask; ///< Bitmask for data/command
#endif                  // end !KINETISK
#else                   // !HAS_PORT_SET_CLR
  ADAGFX_PORT_t csPinMaskSet;     ///< Bitmask for chip select SET (OR)
  ADAGFX_PORT_t csPinMaskClr;     ///< Bitmask for chip select CLEAR (AND)
  ADAGFX_PORT_t dcPinMaskSet;     ///< Bitmask for data/command SET (OR)
  ADAGFX_PORT_t dcPinMaskClr;     ///< Bitmask for data/command CLEAR (AND)
#endif                  // end HAS_PORT_SET_CLR
#endif                  // end USE_FAST_PINIO
  uint8_t connection;      ///< TFT_HARD_SPI, TFT_SOFT_SPI, etc.
  int8_t _rst;             ///< Reset pin # (or -1)
  int8_t _cs;              ///< Chip select pin # (or -1)
  int8_t _dc;              ///< Data/command pin #

  int16_t _xstart = 0;          ///< Internal framebuffer X offset
  int16_t _ystart = 0;          ///< Internal framebuffer Y offset
  uint8_t invertOnCommand = 0;  ///< Command to enable invert mode
  uint8_t invertOffCommand = 0; ///< Command to disable invert mode

  uint32_t _freq = 0; ///< Dummy var to keep subclasses happy
};

class Adafruit_GFX : public Print {

public:
  Adafruit_GFX(int16_t w, int16_t h); // Constructor

  virtual void drawPixel(int16_t x, int16_t y, uint16_t color) = 0;
  virtual void startWrite(void);
  virtual void writePixel(int16_t x, int16_t y, uint16_t color);
  virtual void writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                             uint16_t color);
  virtual void writeFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  virtual void writeFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  virtual void writeLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                         uint16_t color);
  virtual void endWrite(void);
  virtual void setRotation(uint8_t r);
  virtual void invertDisplay(bool i);
  virtual void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  virtual void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  virtual void fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                        uint16_t color);
  virtual void fillScreen(uint16_t color);
  // Optional and probably not necessary to change
  virtual void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                        uint16_t color);
  virtual void drawRect(int16_t x, int16_t y, int16_t w, int16_t h,
                        uint16_t color);

  void drawCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);
  void drawCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
                        uint16_t color);
  void fillCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);
  void fillCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
                        int16_t delta, uint16_t color);
  void drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2,
```

```
                    int16_t y2, uint16_t color);
  void fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2,
                    int16_t y2, uint16_t color);
  void drawRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
                     int16_t radius, uint16_t color);
  void fillRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
                     int16_t radius, uint16_t color);
  void drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                  int16_t h, uint16_t color);
  void drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                  int16_t h, uint16_t color, uint16_t bg);
  void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h,
                  uint16_t color);
  void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h,
                  uint16_t color, uint16_t bg);
  void drawXBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                   int16_t h, uint16_t color);
  void drawGrayscaleBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                           int16_t w, int16_t h);
  void drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w,
                           int16_t h);
  void drawGrayscaleBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                           const uint8_t mask[], int16_t w, int16_t h);
  void drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap, uint8_t *mask,
                           int16_t w, int16_t h);
  void drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[], int16_t w,
                     int16_t h);
  void drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap, int16_t w,
                     int16_t h);
  void drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[],
                     const uint8_t mask[], int16_t w, int16_t h);
  void drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap, uint8_t *mask,
                     int16_t w, int16_t h);
  void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color,
                uint16_t bg, uint8_t size);
  void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color,
                uint16_t bg, uint8_t size_x, uint8_t size_y);
  void getTextBounds(const char *string, int16_t x, int16_t y, int16_t *x1,
                     int16_t *y1, uint16_t *w, uint16_t *h);
  void getTextBounds(const __FlashStringHelper *s, int16_t x, int16_t y,
                     int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h);
  void getTextBounds(const String &str, int16_t x, int16_t y, int16_t *x1,
                     int16_t *y1, uint16_t *w, uint16_t *h);
  void setTextSize(uint8_t s);
  void setTextSize(uint8_t sx, uint8_t sy);
  void setFont(const GFXfont *f = NULL);

  void setCursor(int16_t x, int16_t y) {
    cursor_x = x;
    cursor_y = y;
  }

  void setTextColor(uint16_t c) { textcolor = textbgcolor = c; }

  void setTextColor(uint16_t c, uint16_t bg) {
    textcolor = c;
    textbgcolor = bg;
  }

  void setTextWrap(bool w) { wrap = w; }

  void cp437(bool x = true) { _cp437 = x; }

  using Print::write;
#if ARDUINO >= 100
  virtual size_t write(uint8_t);
#else
  virtual void write(uint8_t);
#endif

  int16_t width(void) const { return _width; };
  int16_t height(void) const { return _height; }

  uint8_t getRotation(void) const { return rotation; }

  int16_t getCursorX(void) const { return cursor_x; }
  int16_t getCursorY(void) const { return cursor_y; };

protected:
  void charBounds(unsigned char c, int16_t *x, int16_t *y, int16_t *minx,
                  int16_t *miny, int16_t *maxx, int16_t *maxy);
  int16_t WIDTH;        ///< This is the 'raw' display width - never changes
  int16_t HEIGHT;       ///< This is the 'raw' display height - never changes
  int16_t _width;       ///< Display width as modified by current rotation
  int16_t _height;      ///< Display height as modified by current rotation
  int16_t cursor_x;     ///< x location to start print()ing text
  int16_t cursor_y;     ///< y location to start print()ing text
  uint16_t textcolor;   ///< 16-bit background color for print()
  uint16_t textbgcolor; ///< 16-bit text color for print()
  uint8_t textsize_x;   ///< Desired magnification in X-axis of text to print()
  uint8_t textsize_y;   ///< Desired magnification in Y-axis of text to print()
  uint8_t rotation;     ///< Display rotation (0 thru 3)
  bool wrap;            ///< If set, 'wrap' text at right edge of display
  bool _cp437;          ///< If set, use correct CP437 charset (default is off)
  GFXfont *gfxFont;     ///< Pointer to special font
};

class Adafruit_GFX_Button {
public:
  Adafruit_GFX_Button(void);
```

```
  void initButton(Adafruit_GFX *gfx, int16_t x, int16_t y, uint16_t w,
                  uint16_t h, uint16_t outline, uint16_t fill,
                  uint16_t textcolor, char *label, uint8_t textsize);
  void initButton(Adafruit_GFX *gfx, int16_t x, int16_t y, uint16_t w,
                  uint16_t h, uint16_t outline, uint16_t fill,
                  uint16_t textcolor, char *label, uint8_t textsize_x,
                  uint8_t textsize_y);
  void initButtonUL(Adafruit_GFX *gfx, int16_t x1, int16_t y1, uint16_t w,
                    uint16_t h, uint16_t outline, uint16_t fill,
                    uint16_t textcolor, char *label, uint8_t textsize);
  void initButtonUL(Adafruit_GFX *gfx, int16_t x1, int16_t y1, uint16_t w,
                    uint16_t h, uint16_t outline, uint16_t fill,
                    uint16_t textcolor, char *label, uint8_t textsize_x,
                    uint8_t textsize_y);
  void drawButton(bool inverted = false);
  bool contains(int16_t x, int16_t y);

  void press(bool p) {
    laststate = currstate;
    currstate = p;
  }

  bool justPressed();
  bool justReleased();

  bool isPressed(void) { return currstate; };

private:
  Adafruit_GFX *_gfx;
  int16_t _x1, _y1; // Coordinates of top-left corner
  uint16_t _w, _h;
  uint8_t _textsize_x;
  uint8_t _textsize_y;
  uint16_t _outlinecolor, _fillcolor, _textcolor;
  char _label[10];

  bool currstate, laststate;
};
```

class GFXcanvas1 : public Adafruit_GFX {

```
public:
  GFXcanvas1(uint16_t w, uint16_t h);
  ~GFXcanvas1(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  bool getPixel(int16_t x, int16_t y) const;
  uint8_t *getBuffer(void) const { return buffer; }

protected:
  bool getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t *buffer; ///< Raster data: no longer private, allow subclass access

private:
};
```

class GFXcanvas8 : public Adafruit_GFX {

```
public:
  GFXcanvas8(uint16_t w, uint16_t h);
  ~GFXcanvas8(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t getPixel(int16_t x, int16_t y) const;
  uint8_t *getBuffer(void) const { return buffer; }

protected:
  uint8_t getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t *buffer; ///< Raster data: no longer private, allow subclass access
};
```

class GFXcanvas16 : public Adafruit_GFX {

```
public:
  GFXcanvas16(uint16_t w, uint16_t h);
  ~GFXcanvas16(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void byteSwap(void);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint16_t getPixel(int16_t x, int16_t y) const;
  uint16_t *getBuffer(void) const { return buffer; }

protected:
  uint16_t getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint16_t *buffer; ///< Raster data: no longer private, allow subclass access
};
```